Phased Array System Toolbox™

Reference

MATLAB® & SIMULINK®

R2023a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

Phased Array System Toolbox™ Reference

Revision History

April 2011	Online only	Revised for version 1.0 (Release 2011a)
September 2011	Online only	Revised for Version 1.1 (R2011b)
March 2012	Online only	Revised for Version 1.2 (R2012a)
September 2012	Online only	Revised for Version 1.3 (R2012b)
March 2013	Online only	Revised for Version 2.0 (R2013a)
September 2013	Online only	Revised for Version 2.1 (R2013b)
March 2014	Online only	Revised for Version 2.2 (R2014a)
October 2014	Online only	Revised for Version 2.3 (R2014b)
March 2015	Online only	Revised for Version 3.0 (R2015a)
September 2015	Online only	Revised for Version 3.1 (R2015b)
March 2016	Online only	Revised for Version 3.2 (R2016a)
September 2016	Online only	Revised for Version 3.3 (R2016b)
March 2017	Online only	Revised for Version 3.4 (R2017a)
September 2017	Online only	Revised for Version 3.5 (R2017b)
March 2018	Online only	Revised for Version 3.6 (R2018a)
September 2018	Online only	Revised for Version 4.0 (R2018b)
March 2019	Online only	Revised for Version 4.1 (R2019a)
September 2019	Online only	Revised for Version 4.2 (R2019b)
March 2020	Online only	Revised for Version 4.3 (R2020a)
September 2020	Online only	Revised for Version 4.4 (R2020b)
March 2021	Online only	Revised for Version 4.5 (R2021a)
September 2021	Online only	Revised for Version 4.6 (R2021b)
March 2022	Online only	Revised for Version 4.7 (R2022a)
September 2022	Online only	Revised for Version 4.8 (R2022b)
March 2023	Online only	Revised for Version 5.0 (R2023a)

Contents

1 Objects

2 Functions

3 Blocks

4 Apps

1

Objects

phased.ADPCACanceller

Package: phased

Adaptive DPCA (ADPCA) pulse canceller

Description

The ADPCACanceller object implements an adaptive displaced phase center array pulse canceller for a uniform linear array (ULA).

To compute the output signal of the space time pulse canceller:

1. Define and set up your ADPCA pulse canceller. See "Construction" on page 1-2.
2. Call step to execute the ADPCA algorithm according to the properties of phased.ADPCACanceller. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object™, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.ADPCACanceller creates an adaptive displaced phase center array (ADPCA) canceller System object, H. This object performs two-pulse ADPCA processing on the input data.

H = phased.ADPCACanceller(Name,Value) creates an ADPCA object, H, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN). See "Properties" on page 1-2 for the list of available property names.

Properties

SensorArray

Uniform linear array

Uniform linear array, specified as a phased.ULA System object.

Default: phased.ULA with default property values

PropagationSpeed

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

PRFSource

Source of pulse repetition frequency

Source of the PRF values for the STAP processor, specified as `'Property'` or `'Input port'`. When you set this property to `'Property''`, the PRF is determined by the value of the `PRF` property. When you set this property to `'Input port'`, the PRF is determined by an input argument to the `step` method at execution time.

Default: `'Property'`

PRF

Pulse repetition frequency

Pulse repetition frequency (PRF) of the received signal, specified as a positive scalar. Units are in Hertz. This property can be specified as single or double precision.

Dependencies

To enable this property, set the `PRFSource` property to `'Property'`.

Default: `1`

DirectionSource

Source of receiving main lobe direction

Specify whether the targeting direction for the STAP processor comes from the `Direction` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `Direction` property of this object specifies the targeting direction.
`'Input port'`	An input argument in each invocation of `step` specifies the targeting direction.

Default: `'Property'`

Direction

Receiving mainlobe direction (degrees)

Specify the receiving mainlobe direction of the receiving sensor array as a column vector of length 2. The direction is specified in the format of `[AzimuthAngle; ElevationAngle]` (in degrees). Azimuth angle should be between –180 and 180. Elevation angle should be between –90 and 90. This property applies when you set the `DirectionSource` property to `'Property'`. This property can be specified as single or double precision.

Default: `[0; 0]`

NumPhaseShifterBits

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed. You can specify this property as single or double precision.

Default: `0`

DopplerSource

Source of targeting Doppler

Specify whether the targeting Doppler for the STAP processor comes from the `Doppler` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `Doppler` property of this object specifies the Doppler.
`'Input port'`	An input argument in each invocation of `step` specifies the Doppler.

Default: `'Property'`

Doppler

Targeting Doppler frequency (Hz)

Specify the targeting Doppler of the STAP processor as a scalar. This property applies when you set the `DopplerSource` property to `'Property'`. This property can be specified as single or double precision.

Default: `0`

WeightsOutputPort

Output processing weights

To obtain the weights used in the STAP processor, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the weights, set this property to `false`.

Default: `false`

PreDopplerOutput

Output pre-Doppler result

Set this property to `true` to output the processing result before applying the Doppler filtering. Set this property to `false` to output the processing result after the Doppler filtering.

Default: `false`

NumGuardCells

Number of guard cells

Specify the number of guard cells used in the training as an even integer. This property specifies the total number of cells on both sides of the cell under test. This property can be specified as single or double precision.

Default: 2, indicating that there is one guard cell at both the front and back of the cell under test

`NumTrainingCells`

Number of training cells

Specify the number of training cells used in the training as an even integer. Whenever possible, the training cells are equally divided before and after the cell under test. This property can be specified as single or double precision.

Default: 2, indicating that there is one training cell at both the front and back of the cell under test

Methods

step	Perform ADPCA processing on input data

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Process Radar Data Cube Using ADPCA Processor

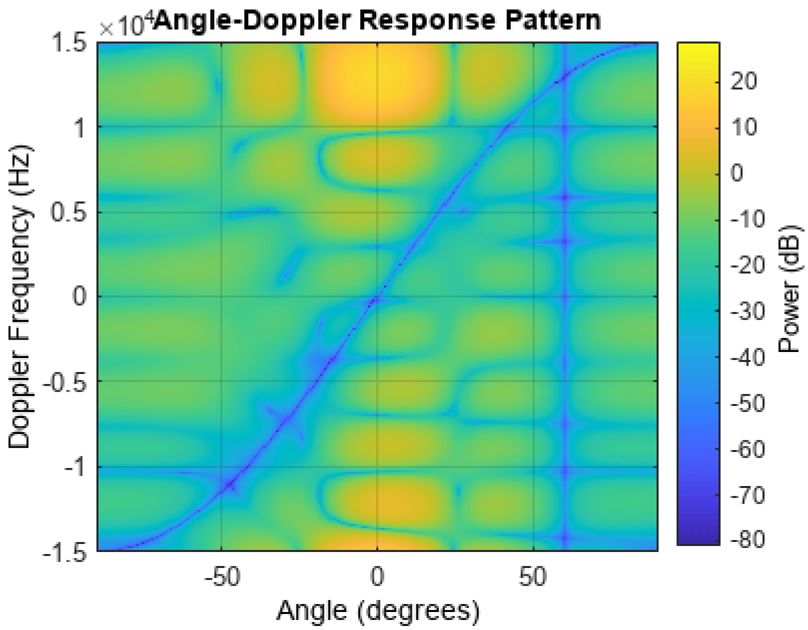
Process a radar data cube using an ADPCA processor. Weights are calculated for the 71st cell of the data cube. Set the look direction to (0,0) degrees and the Doppler shift to 12.980 kHz.

Load radar data file and compute weights

```
load STAPExampleData;
canceller = phased.ADPCACanceller('SensorArray',STAPEx_HArray,...
    'PRF',STAPEx_PRF,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'NumTrainingCells',100,...
    'WeightsOutputPort',true,...
    'DirectionSource','Input port',...
    'DopplerSource','Input port');
[y,w] = canceller(STAPEx_ReceivePulse,71,[0; 0],12.980e3);
```

Create AnglerDoppler System object and plot response

```
sAngeDop = phased.AngleDopplerResponse(...
    'SensorArray',canceller.SensorArray,...
    'OperatingFrequency',canceller.OperatingFrequency,...
    'PRF',canceller.PRF,...
    'PropagationSpeed',canceller.PropagationSpeed);
plotResponse(sAngeDop,w)
```



Algorithms

Single Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Guerci, J. R. *Space-Time Adaptive Processing for Radar*. Boston: Artech House, 2003.

[2] Ward, J. “Space-Time Adaptive Processing for Airborne Radar Data Systems,” *Technical Report 1015*, MIT Lincoln Laboratory, December, 1994.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.AngleDopplerResponse` | `phased.DPCACanceller` | `phased.STAPSMIBeamformer` | `uv2azel` | `phitheta2azel`

step

System object: `phased.ADPCACanceller`
Package: `phased`

Perform ADPCA processing on input data

Syntax

```
Y = step(H,X,CUTIDX)
Y = step(H,X,CUTIDX,ANG)
Y = step(H,X,CUTIDX,DOP)
Y = step(H,X,CUTIDX,PRF)
[Y,W] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X,CUTIDX)` applies the ADPCA pulse cancellation algorithm to the input data `X`. The algorithm calculates the processing weights according to the range cell specified by `CUTIDX`. This syntax is available when the `DirectionSource` property is `'Property'` and the `DopplerSource` property is `'Property'`. The receiving mainlobe direction is the `Direction` property value. The output `Y` contains the result of pulse cancellation either before or after Doppler filtering, depending on the `PreDopplerOutput` property value.

`Y = step(H,X,CUTIDX,ANG)` uses `ANG` as the receiving main lobe direction. This syntax is available when the `DirectionSource` property is `'Input port'` and the `DopplerSource` property is `'Property'`.

`Y = step(H,X,CUTIDX,DOP)` uses `DOP` as the targeting Doppler frequency. This syntax is available when the `DopplerSource` property is `'Input port'`.

`Y = step(H,X,CUTIDX,PRF)` uses `PRF` as the pulse repetition frequency. This syntax is available when the `PRFSource` property is `'Input port'`.

`[Y,W] = step( ___ )` also returns the processing weights, `W`. This syntax is available when the `WeightsOutputPort` property is `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Pulse canceller object.

X

Input data. X must be a 3-dimensional M-by-N-by-P numeric array whose dimensions are (range, channels, pulses). You can specify this argument as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

CUTIDX

Range cell. You can specify this argument as single or double precision.

PRF

Pulse repetition frequency specified as a positive scalar. To enable this argument, set the PRFSource property to 'Input port'. You can specify this argument as single or double precision. Units are in Hertz.

ANG

Receiving main lobe direction. ANG must be a 2-by-1 vector in the form [AzimuthAngle; ElevationAngle], in degrees. The azimuth angle must be between –180 and 180. The elevation angle must be between –90 and 90. You can specify this argument as single or double precision.

Default: Direction property of H

DOP

Targeting Doppler frequency in hertz. DOP must be a scalar. You can specify this argument as single or double precision.

Default: Doppler property of H

Output Arguments

Y

Result of applying pulse cancelling to the input data. The meaning and dimensions of Y depend on the PreDopplerOutput property of H:

- If PreDopplerOutput is true, Y contains the pre-Doppler data. Y is an M-by-(P–1) matrix. Each column in Y represents the result obtained by cancelling the two successive pulses.
- If PreDopplerOutput is false, Y contains the result of applying an FFT-based Doppler filter to the pre-Doppler data. The targeting Doppler is the Doppler property value. Y is a column vector of length M.

W

Processing weights the pulse canceller used to obtain the pre-Doppler data. The dimensions of W depend on the PreDopplerOutput property of H:

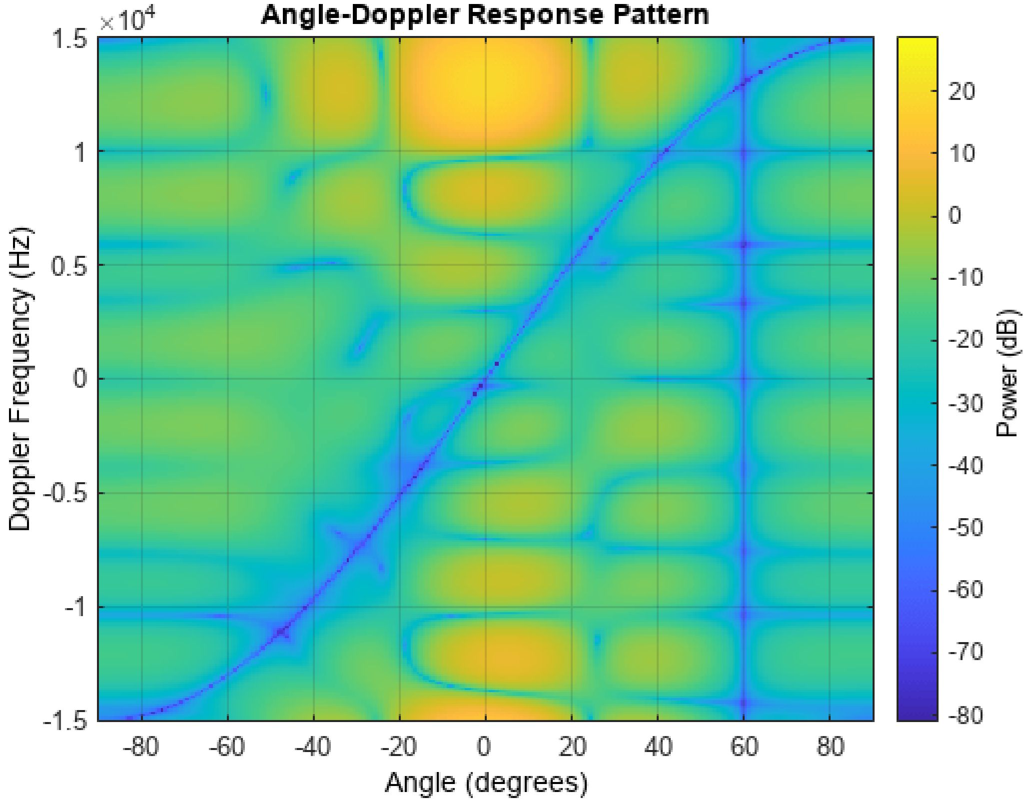
- If PreDopplerOutput is true, W is a 2N-by-(P-1) matrix. The columns in W correspond to successive pulses in X.
- If PreDopplerOutput is false, W is a column vector of length (N*P).

Examples

Plot Response of ADPCA Processor with Quantized Weights

Process a radar data cube using an ADPCA processor. Weights are calculated for the 71st cell of the data cube. Load the data cube from STAPExampleData.mat. Quantize the weights to 4 bits. Set the look direction to (0,0) degrees and the Doppler shift to 12.980 kHz.

```
load STAPExampleData;
sADPCA = phased.ADPCACanceller('SensorArray',STAPEx_HArray,...
    'PRF',STAPEx_PRF,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'NumTrainingCells',100,...
    'WeightsOutputPort',true,...
    'DirectionSource','Input port',...
    'DopplerSource','Input port',...
    'NumPhaseShifterBits',4);
[y,w] = step(sADPCA,STAPEx_ReceivePulse,71,[0; 0],12.980e3);
sAngDop = phased.AngleDopplerResponse(...
    'SensorArray',sADPCA.SensorArray,...
    'OperatingFrequency',sADPCA.OperatingFrequency,...
    'PRF',sADPCA.PRF,...
    'PropagationSpeed',sADPCA.PropagationSpeed);
plotResponse(sAngDop,w);
```



See Also

`uv2azel` | `phitheta2azel`

phased.AngleDopplerResponse

Package: phased

Angle-Doppler response

Description

The AngleDopplerResponse object calculates the angle-Doppler response of input data.

To compute the angle-Doppler response:

1. Define and set up your angle-Doppler response calculator. See "Construction" on page 1-12.
2. Call step to compute the angle-Doppler response of the input signal according to the properties of phased.AngleDopplerResponse. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.AngleDopplerResponse creates an angle-Doppler response System object, H. This object calculates the angle-Doppler response of the input data.

H = phased.AngleDopplerResponse(Name,Value) creates angle-Doppler object, H, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

SensorArray

Sensor array

Sensor array specified as an array System object belonging to the phased package. A sensor array can contain subarrays.

Default: phased.ULA with default property values

PropagationSpeed

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

`OperatingFrequency`

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

`PRFSource`

Source of PRF values

Source of the PRF values for the STAP processor, specified as `'Property'` or `'Input port'`. When you set this property to `'Property'`, the PRF is determined by the value of the `PRF` property. When you set this property to `'Input port'`, the PRF is determined by an input argument to the `step` method at execution time.

Default: `'Property'`

`PRF`

Pulse repetition frequency

Specify the pulse repetition frequency (PRF) in hertz of the input signal as a positive scalar. This property applies when you set the `PRFSource` property to `'Property'`. You can specify this property as single or double precision.

Default: `1`

`ElevationAngleSource`

Source of elevation angle

Specify whether the elevation angle comes from the `ElevationAngle` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `ElevationAngle` property of this object specifies the elevation angle.
`'Input port'`	An input argument in each invocation of `step` specifies the elevation angle.

Default: `'Property'`

`ElevationAngle`

Elevation angle

Specify the elevation angle in degrees used to calculate the angle-Doppler response as a scalar. The angle must be between –90 and 90. This property applies when you set the `ElevationAngleSource` property to `'Property'`. You can specify this property as single or double precision.

Default: `0`

`NumAngleSamples`

Number of samples in angular domain

Specify the number of samples in the angular domain used to calculate the angle-Doppler response as a positive integer. This value must be greater than 2. You can specify this property as single or double precision.

Default: 256

`NumDopplerSamples`

Number of samples in Doppler domain

Specify the number of samples in the Doppler domain used to calculate the angle-Doppler response as a positive integer. This value must be greater than 2. You can specify this property as single or double precision.

Default: 256

Methods

plotResponse	Plot angle-Doppler response
step	Calculate angle-Doppler response

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Calculate Angle-Doppler Response

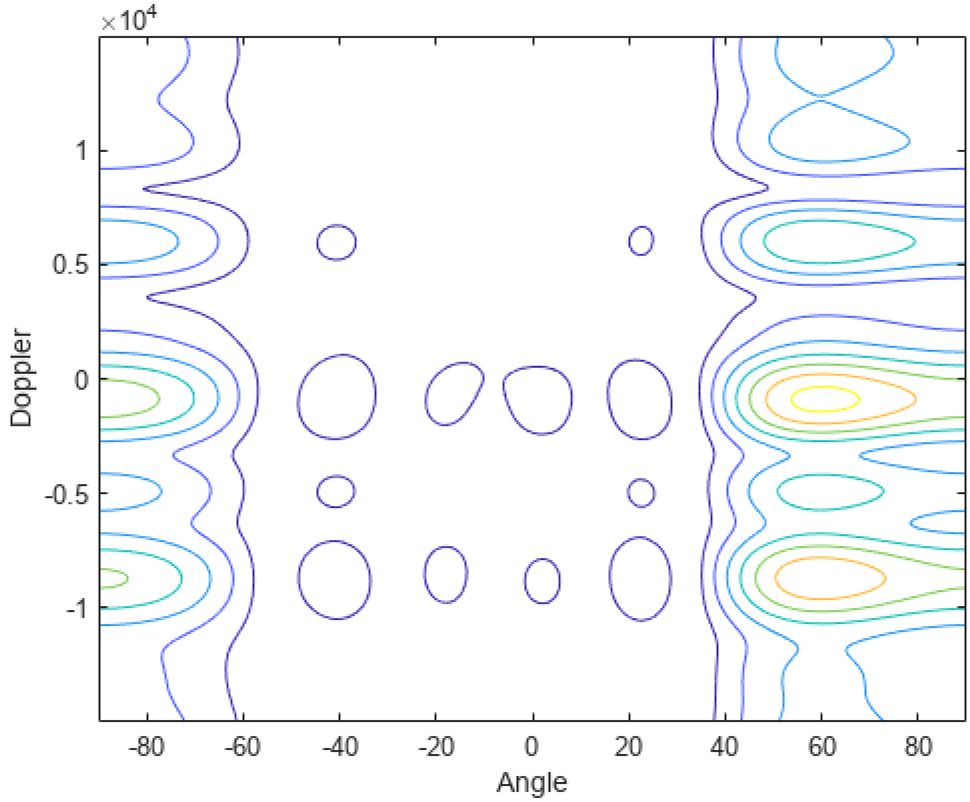
Calculate the angle-Doppler response of the 190th cell of a collected data cube.

Load data cube and construct a `phased.AngleDopplerResponse` System object™.

```
load STAPExampleData;
x = shiftdim(STAPEx_ReceivePulse(190,:,:));
response = phased.AngleDopplerResponse(...
    'SensorArray',STAPEx_HArray,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'PRF',STAPEx_PRF);
```

Plot angle-Doppler response.

```
[resp,ang_grid,dop_grid] = response(x);
contour(ang_grid,dop_grid,abs(resp))
xlabel('Angle')
ylabel('Doppler')
```



Algorithms

Response Computation

`phased.AngleDopplerResponse` generates the response using a conventional beamformer and an FFT-based Doppler filter. For further details, see [1].

Single Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Guerci, J. R. *Space-Time Adaptive Processing for Radar*. Boston: Artech House, 2003.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.ADPCACanceller` | `phased.DPCACanceller` | `phased.STAPSMIBeamformer` | `uv2azel` | `phitheta2azel`

plotResponse

System object: phased.AngleDopplerResponse
Package: phased

Plot angle-Doppler response

Syntax

```
plotResponse(H,X)
plotResponse(H,X,ELANG)
plotResponse(H,X,PRF)
plotResponse( ___ ,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,X)` plots the angle-Doppler response of the data in `X` in decibels. This syntax is available when the `ElevationAngleSource` property is `'Property'`.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`plotResponse(H,X,ELANG)` plots the angle-Doppler response calculated using the specified elevation angle `ELANG`. This syntax is available when the `ElevationAngleSource` property is `'Input port'`.

`plotResponse(H,X,PRF)` plots the angle-Doppler response calculated using the specified pulse repetition frequency `PRF`. This syntax is available when the `PRFSource` property is `'Input port'`.

`plotResponse( ___ ,Name,Value)` plots the angle-Doppler response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns the handle of the image in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Angle-Doppler response object.

X

Input data.

ELANG

Elevation angle in degrees.

Default: Value of `Elevation` property of `H`

PRF

Pulse repetition frequency specified as a positive scalar. To enable this argument, set the `PRFSource` property to `'Input port'`. Units are in Hertz.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

NormalizeDoppler

Set this value to `true` to normalize the Doppler frequency. Set this value to `false` to plot the angle-Doppler response without normalizing the Doppler frequency.

Default: `false`

Unit

The unit of the plot. Valid values are `'db'`, `'mag'`, and `'pow'`.

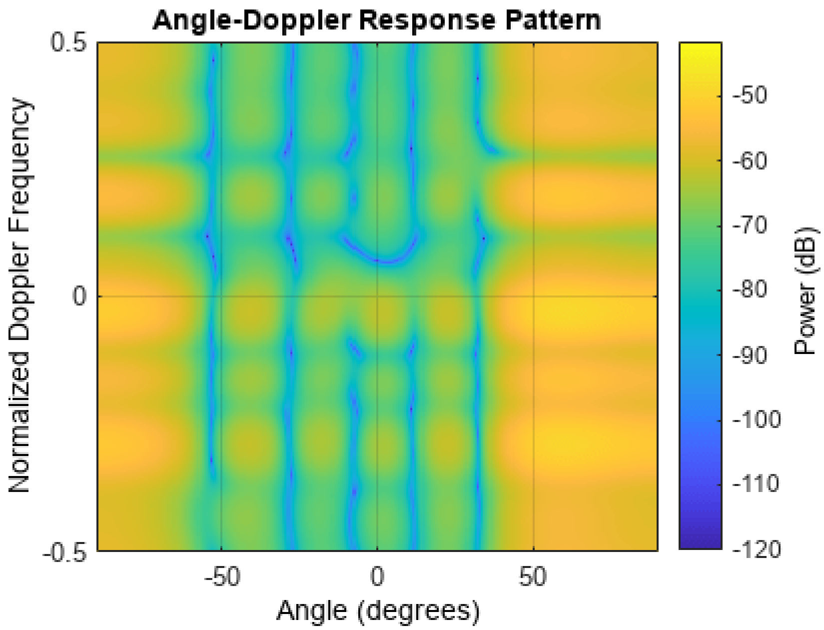
Default: `'db'`

Examples

Plot Angle-Doppler Response

Plot the angle-Doppler response of the 190th cell of a collected data cube.

```
load STAPExampleData;
x = shiftdim(STAPEx_ReceivePulse(190,:,:));
hadresp = phased.AngleDopplerResponse(...
    'SensorArray',STAPEx_HArray,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'PRF',STAPEx_PRF);
plotResponse(hadresp,x,'NormalizeDoppler',true);
```



See Also

`uv2azel` | `phitheta2azel`

step

System object: phased.AngleDopplerResponse
Package: phased

Calculate angle-Doppler response

Syntax

```
[RESP,ANG_GRID,DOP_GRID] = step(H,X)
[RESP,ANG_GRID,DOP_GRID] = step(H,X,ELANG)
RESP,ANG_GRID,DOP_GRID = step(H,X,PRF)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[RESP,ANG_GRID,DOP_GRID] = step(H,X)` calculates the angle-Doppler response of the data `X`. `RESP` is the complex angle-Doppler response. `ANG_GRID` and `DOP_GRID` provide the angle samples and Doppler samples, respectively, at which the angle-Doppler response is evaluated. This syntax is available when the `ElevationAngleSource` property is `'Property'`.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`[RESP,ANG_GRID,DOP_GRID] = step(H,X,ELANG)` calculates the angle-Doppler response using the specified elevation angle `ELANG`. This syntax is available when the `ElevationAngleSource` property is `'Input port'`.

`RESP,ANG_GRID,DOP_GRID = step(H,X,PRF)` uses `PRF` as the pulse repetition frequency. This syntax is available when the `PRFSource` property is `'Input port'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Angle-Doppler response object.

X

Input data as a matrix or column vector.

If X is a matrix, the number of rows in the matrix must equal the number of elements of the array specified in the SensorArray property of H.

If X is a vector, the number of rows must be an integer multiple of the number of elements of the array specified in the SensorArray property of H. In addition, the multiple must be at least 2.

ELANG

Elevation angle in degrees. You can specify this argument as single or double precision.

Default: Value of Elevation property of H

PRF

Pulse repetition frequency specified as a positive scalar. To enable this argument, set the PRFSource property to 'Input port'. Units are in Hertz. You can specify this argument as single or double precision.

Output Arguments

RESP

Complex angle-Doppler response of X. RESP is a P-by-Q matrix. P is determined by the NumDopplerSamples property of H and Q is determined by the NumAngleSamples property.

ANG_GRID

Angle samples at which the angle-Doppler response is evaluated. ANG_GRID is a column vector of length Q.

DOP_GRID

Doppler samples at which the angle-Doppler response is evaluated. DOP_GRID is a column vector of length P.

Examples

Calculate Angle-Doppler Response

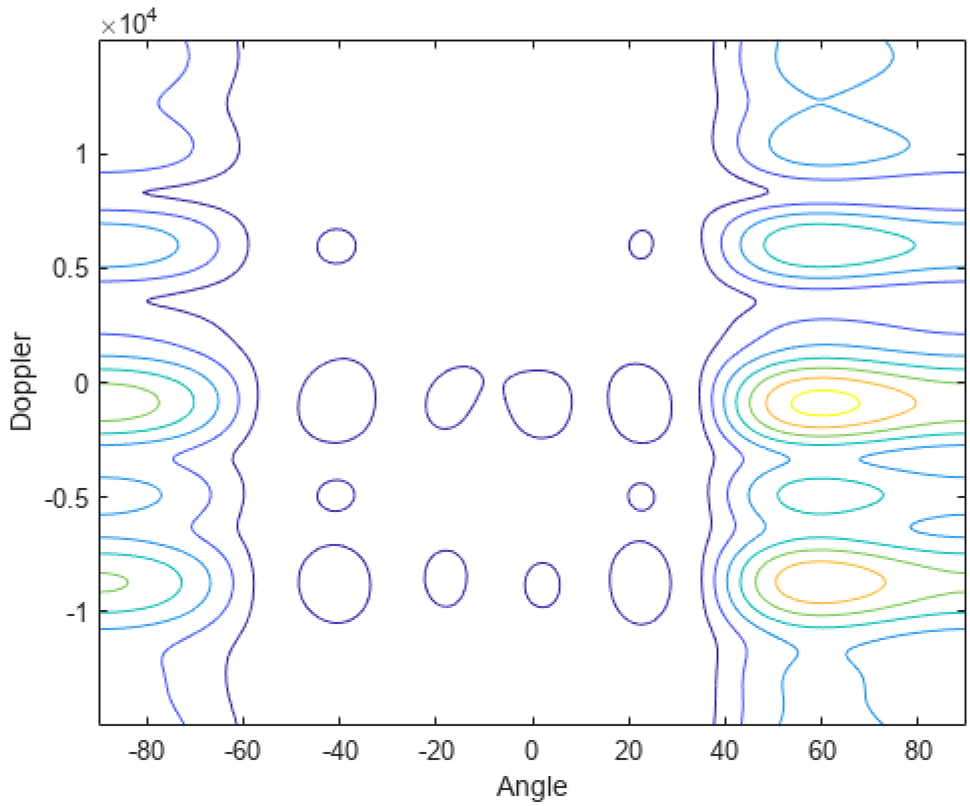
Calculate the angle-Doppler response of the 190th cell of a collected data cube.

Load data cube and construct a phased.AngleDopplerResponse System object™.

```
load STAPExampleData;
x = shiftdim(STAPEx_ReceivePulse(190,:,:));
response = phased.AngleDopplerResponse(...
    'SensorArray',STAPEx_HArray,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'PRF',STAPEx_PRF);
```

Plot angle-Doppler response.

```
[resp,ang_grid,dop_grid] = response(x);
contour(ang_grid,dop_grid,abs(resp))
xlabel('Angle')
ylabel('Doppler')
```



Algorithms

`phased.AngleDopplerResponse` generates the response using a conventional beamformer and an FFT-based Doppler filter. For further details, see [1].

References

[1] Guerci, J. R. *Space-Time Adaptive Processing for Radar*. Boston: Artech House, 2003.

See Also

`uv2azel` | `phitheta2azel` | `azel2uv` | `azel2phitheta`

phased.ArrayGain

Package: `phased`

Sensor array gain

Description

The `ArrayGain` object calculates the array gain for a sensor array. The array gain on page 1-24 is defined as the signal to noise ratio (SNR) improvement between the array output and the individual channel input, assuming the noise is spatially white. It is related to the array response but is not the same.

To compute the SNR gain of the antenna for specified directions:

1. Define and set up your array gain calculator. See "Construction" on page 1-23.
2. Call `step` to estimate the gain according to the properties of `phased.ArrayGain`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.ArrayGain` creates an array gain System object, `H`. This object calculates the array gain of a 2-element uniform linear array for specified directions.

`H = phased.ArrayGain(Name,Value)` creates and array-gain object, `H`, with the specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Sensor array

Sensor array specified as an array System object belonging to the `phased` package. A sensor array can contain subarrays.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar.

Default: Speed of light

WeightsInputPort

Add input to specify weights

To specify weights, set this property to `true` and use the corresponding input argument when you invoke `step`. If you do not want to specify weights, set this property to `false`.

Default: `false`

Methods

step	Calculate array gain of sensor array

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Array Gain of 4-Element ULA

Calculate the array gain for a 4-element uniform linear array (ULA) in the direction 30° azimuth and 20° elevation. The array operating frequency is 300 MHz.

```
fc = 300e6;
array = phased.ULA(4);
gain = phased.ArrayGain('SensorArray',array);
g = gain(fc,[30;20])
```

```
g = -17.1783
```

More About

Array Gain

The array gain is defined as the signal to noise ratio (SNR) improvement between the array output and the individual channel input, assuming the noise is spatially white. You can express the array gain as follows:

$$\frac{SNR_{\text{out}}}{SNR_{\text{in}}} = \frac{\left(\frac{w^H v s v^H w}{w^H N w}\right)}{\left(\frac{s}{N}\right)} = \frac{w^H v v^H w}{w^H w}$$

In this equation:

- w is the vector of weights applied on the sensor array. When you use `phased.ArrayGain`, you can optionally specify weights by setting the `WeightsInputPort` property to `true` and specifying the `W` argument in the `step` method syntax.
- v is the steering vector representing the array response toward a given direction. When you call the `step` method, the `ANG` argument specifies the direction.

- s is the input signal power.
- N is the noise power.
- H denotes the complex conjugate transpose.

For example, if a rectangular taper is used in the array, the array gain is the square of the array response normalized by the number of elements in the array.

Version History

Introduced in R2011a

References

[1] Guerci, J. R. *Space-Time Adaptive Processing for Radar*. Boston: Artech House, 2003.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support arrays containing polarized antenna elements, that is, the `phased.ShortDipoleAntennaElement` or `phased.CrossedDipoleAntennaElement` antennas.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.ArrayResponse` | `phased.ElementDelay` | `phased.SteeringVector`

step

System object: `phased.ArrayGain`
Package: `phased`

Calculate array gain of sensor array

Syntax

```
G = step(H,FREQ,ANG)
G = step(H,FREQ,ANG,WEIGHTS)
G = step(H,FREQ,ANG,STEERANGLE)
G = step(H,FREQ,ANG,WEIGHTS,STEERANGLE)
G = step(H,FREQ,ANG,WS)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`G = step(H,FREQ,ANG)` returns the array gain on page 1-29 `G` of the array for the operating frequencies specified in `FREQ` and directions specified in `ANG`.

`G = step(H,FREQ,ANG,WEIGHTS)` applies weights `WEIGHTS` on the sensor array. This syntax is available when you set the `WeightsInputPort` property to `true`.

`G = step(H,FREQ,ANG,STEERANGLE)` uses `STEERANGLE` as the subarray steering angle. This syntax is available when you configure `H` so that `H.Sensor` is an array that contains subarrays, and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

`G = step(H,FREQ,ANG,WEIGHTS,STEERANGLE)` combines all input arguments. This syntax is available when you configure `H` so that `H.WeightsInputPort` is `true`, `H.Sensor` is an array that contains subarrays, and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

`G = step(H,FREQ,ANG,WS)` uses `WS` as weights applied to each element within each subarray. To use this syntax, set the `SensorArray` property to an array that supports subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Array gain object.

FREQ

Operating frequencies of array in hertz. FREQ is a row vector of length L. Typical values are within the range specified by a property of the sensor element. The element is H.SensorArray.Element, H.SensorArray.Array.Element, or H.SensorArray.Subarray.Element, depending on the type of array. The frequency range property is named FrequencyRange or FrequencyVector, depending on the type of element in the array. The element has zero response at frequencies outside that range.

ANG

Directions in degrees. ANG can be either a 2-by-M matrix or a row vector of length M.

If ANG is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If ANG is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

WEIGHTS

Weights on the sensor array. WEIGHTS can be either an N-by-L matrix or a column vector of length N. N is the number of subarrays if H.SensorArray contains subarrays, or the number of elements otherwise. L is the number of frequencies specified in FREQ.

If WEIGHTS is a matrix, each column of the matrix represents the weights at the corresponding frequency in FREQ.

If WEIGHTS is a vector, the weights apply at all frequencies in FREQ.

STEERANGLE

Subarray steering angle in degrees. STEERANGLE can be a length-2 column vector or a scalar.

If STEERANGLE is a length-2 vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, and the elevation angle must be between –90 and 90 degrees.

If STEERANGLE is a scalar, it represents the azimuth angle. In this case, the elevation angle is assumed to be 0.

WS

Subarray element weights

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray element weights

Sensor Array	Subarray weights
`phased.ReplicatedSubarray`	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of `WS` specifies the weights for the corresponding subarray.
`phased.PartitionedArray`	Subarrays may not have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `SensorArray` property to an array that contains subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Output Arguments

G

Gain of sensor array, in decibels. `G` is an M-by-L matrix. `G` contains the gain at the M angles specified in `ANG` and the L frequencies specified in `FREQ`.

Examples

Array Gain of 6-Element ULA

Construct a uniform linear array (ULA) having six elements and operating at 1 GHz. The array elements are spaced at one-half the operating wavelength. Find the array gain in dB in the direction 45° azimuth and 10° elevation.

Create the `phased.ArrayGain` System object™.

```
fc = 1e9;
lambda = physconst('LightSpeed')/fc;
array = phased.ULA('NumElements',6,'ElementSpacing',lambda/2);
gain = phased.ArrayGain('SensorArray',array);
```

Determine array gain at the specified operating frequency and angle.

```
arraygain = gain(fc,[45;10])
```

```
arraygain = -17.9275
```

More About

Array Gain

The array gain is defined as the signal to noise ratio (SNR) improvement between the array output and the individual channel input, assuming the noise is spatially white. You can express the array gain as follows:

$$\frac{SNR_{\text{out}}}{SNR_{\text{in}}} = \frac{\left(\frac{w^H v s v^H w}{w^H N w}\right)}{\left(\frac{s}{N}\right)} = \frac{w^H v v^H w}{w^H w}$$

In this equation:

- w is the vector of weights applied on the sensor array. When you use `phased.ArrayGain`, you can optionally specify weights by setting the `WeightsInputPort` property to `true` and specifying the `W` argument in the `step` method syntax.
- v is the steering vector representing the array response toward a given direction. When you call the `step` method, the `ANG` argument specifies the direction.
- s is the input signal power.
- N is the noise power.
- H denotes the complex conjugate transpose.

For example, if a rectangular taper is used in the array, the array gain is the square of the array response normalized by the number of elements in the array.

See Also

`uv2azel` | `phitheta2azel`

phased.ArrayResponse

Package: `phased`

Sensor array response

Description

The `ArrayResponse` object calculates the complex-valued response of a sensor array.

To compute the response of the array for specified directions:

1. Define and set up your array response calculator. See "Construction" on page 1-30.
2. Call `step` to estimate the response according to the properties of `phased.ArrayResponse`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.ArrayResponse` creates an array response System object, `H`. This object calculates the response of a sensor array for the specified directions. By default, a 2-element uniform linear array (ULA) is used.

`H = phased.ArrayResponse(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array used to calculate response

Specify the sensor array as a handle. The sensor array must be an array object in the `phased` package. The array can contain subarrays.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar.

Default: Speed of light

WeightsInputPort

Add input to specify weights

To specify weights, set this property to `true` and use the corresponding input argument when you invoke `step`. If you do not want to specify weights, set this property to `false`.

Default: `false`

EnablePolarization

Enable polarization simulation

Set this property to `true` to let the array response simulate polarization. Set this property to `false` to ignore polarization. This property applies only when the array specified in the `SensorArray` property is capable of simulating polarization.

Default: `false`

Methods

step	Calculate array response of sensor array

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Plot Array Response

Calculate array response for a 4-element uniform linear array (ULA) in the direction of 30 degrees azimuth and 20 degrees elevation. Assume the array's operating frequency is 300 MHz.

Construct ULA and ArrayResponse System objects

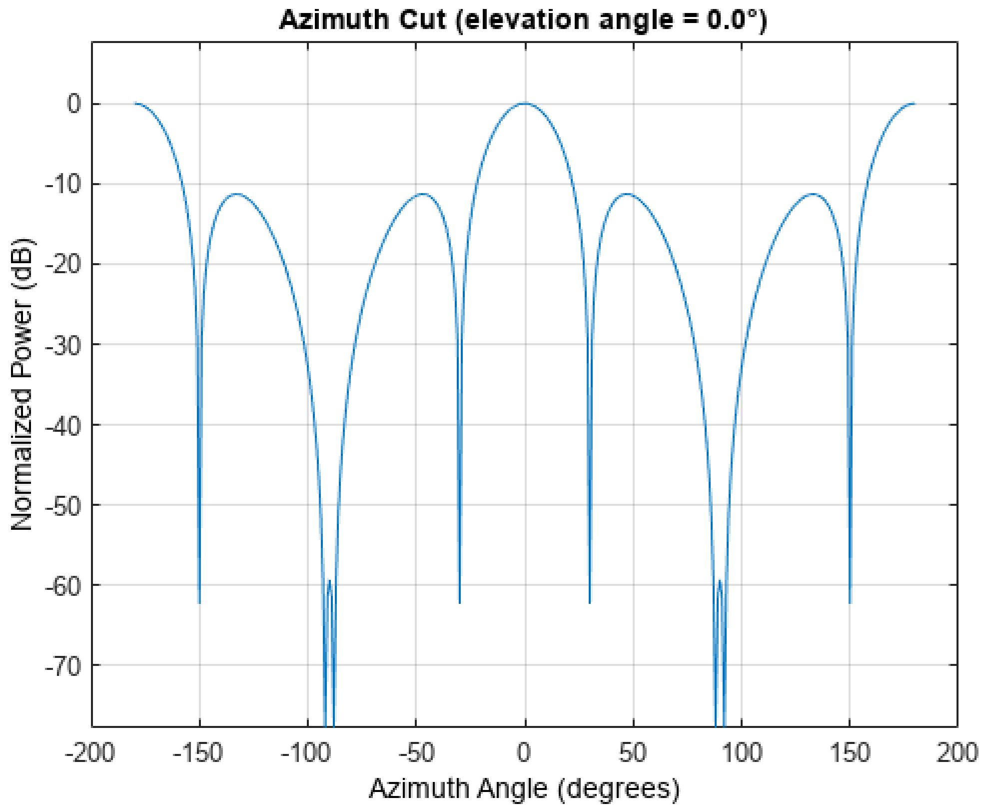
```
fc = 300e6;
c = physconst('LightSpeed');
array = phased.ULA(4);
response = phased.ArrayResponse('SensorArray',array);
resp = response(fc,[30;20])
```

```
resp = 0.2768 + 0.0000i
```

Plot the array response in dB

Plot the normalized power in db as an azimuth cut at 0 degrees elevation.

```
pattern(array,fc,[-180:180],0,'PropagationSpeed',c,'CoordinateSystem','rectangular','Type','power
```



Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.ArrayGain` | `phased.ElementDelay` | `phased.ConformalArray` | `phased.ULA` | `phased.URA` | `phased.SteeringVector`

step

System object: `phased.ArrayResponse`
Package: `phased`

Calculate array response of sensor array

Syntax

```
RESP = step(H,FREQ,ANG)
RESP = step(H,FREQ,ANG,WEIGHTS)
RESP = step(H,FREQ,ANG,STEERANGLE)
RESP = step(H,FREQ,ANG,WEIGHTS,STEERANGLE)
RESP = step(H,FREQ,ANG,WS)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the array response `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`.

`RESP = step(H,FREQ,ANG,WEIGHTS)` applies weights `WEIGHTS` on the sensor array. This syntax is available when you set the `WeightsInputPort` property to `true`.

`RESP = step(H,FREQ,ANG,STEERANGLE)` uses `STEERANGLE` as the subarray steering angle. This syntax is available when you configure `H` so that `H.Sensor` is an array that contains subarrays, and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

`RESP = step(H,FREQ,ANG,WEIGHTS,STEERANGLE)` combines all input arguments. This syntax is available when you configure `H` so that `H.WeightsInputPort` is `true`, `H.Sensor` is an array that contains subarrays, and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

`RESP = step(H,FREQ,ANG,WS)` uses `WS` as weights applied to each element within each subarray. To use this syntax, set the `SensorArray` property to an array that supports subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Array response object.

FREQ

Operating frequencies of array in hertz. FREQ is a row vector of length L. Typical values are within the range specified by a property of the sensor element. The element is H.SensorArray.Element, H.SensorArray.Array.Element, or H.SensorArray.Subarray.Element, depending on the type of array. The frequency range property is named FrequencyRange or FrequencyVector, depending on the type of element in the array. The element has zero response at frequencies outside that range. The element has zero response at frequencies outside that range.

ANG

Directions in degrees. ANG can be either a 2-by-M matrix or a row vector of length M.

If ANG is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If ANG is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

WEIGHTS

Weights on the sensor array. WEIGHTS can be either an N-by-L matrix or a column vector of length N. N is the number of subarrays if H.SensorArray contains subarrays, or the number of elements otherwise. L is the number of frequencies specified in FREQ.

If WEIGHTS is a matrix, each column of the matrix represents the weights at the corresponding frequency in FREQ.

If WEIGHTS is a vector, the weights apply at all frequencies in FREQ.

STEERANGLE

Subarray steering angle in degrees. STEERANGLE can be a length-2 column vector or a scalar.

If STEERANGLE is a length-2 vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, and the elevation angle must be between –90 and 90 degrees.

If STEERANGLE is a scalar, it represents the azimuth angle. In this case, the elevation angle is assumed to be 0.

WS

Subarray element weights

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray element weights

Sensor Array	Subarray weights
phased.ReplicatedSubarray	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of WS specifies the weights for the corresponding subarray.
phased.PartitionedArray	Subarrays may not have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `SensorArray` property to an array that contains subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Output Arguments

RESP

Voltage response of the sensor array. The response depends on whether the `EnablePolarization` property is set to `true` or `false`.

- If the `EnablePolarization` property is set to `false`, the voltage response, `RESP`, has the dimensions M-by-L. M represents the number of angles specified in the input argument `ANG` while L represents the number of frequencies specified in `FREQ`.
- If the `EnablePolarization` property is set to `true`, the voltage response, `RESP`, is a MATLAB® `struct` containing two fields, `RESP.H` and `RESP.V`. The `RESP.H` field represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions M-by-L. M represents the number of angles specified in the input argument, `ANG`, while L represents the number of frequencies specified in `FREQ`.

Examples

Array Response of ULA

Find the response of a 6-element uniform linear array operating at 1 GHz. The array elements are spaced one-half wavelength apart. The incident signal direction is 45° azimuth and 10° elevation. Obtain the response at this direction.

```
fc = 1e9;
lambda = physconst('LightSpeed')/fc;
```

Create the ULA array.

```
array = phased.ULA('NumElements',6,'ElementSpacing',lambda/2);
```

Create the array response System object™.

```
response = phased.ArrayResponse('SensorArray',array);
resp = response(fc,[45;10]);
```

See Also

uv2azel | phitheta2azel

clone

Create identical object

Syntax

```
object_clone = clone(original_object)
```

Description

`object_clone = clone(original_object)` creates a copy, `object_clone`, of the input object, `original_object`, with identical property values.

Input Arguments

`original_object` — Object to be cloned
object

Object to be cloned.

Output Arguments

`object_clone` — Object clone
object

Object clone, returned as an object of the same class as `original_object`.

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

reset

Reset object state and property values

Syntax

```
reset(obj)
```

Description

`reset(obj)` resets the internal state and input properties of the object `obj`.

- If `obj` writes or reads a file, `reset` resets the object to the beginning of the file.
- If `obj` changes properties, `reset` resets the properties to their initial default values.
- If `obj` uses a random number generation seed, `reset` resets the seed property.

Input Arguments

`obj` — Object to reset
object

Object whose state you want to reset.

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

release

Release resources and allow changes to object property values and input characteristics

Syntax

```
release(obj)
```

Description

`release(obj)` releases system resources such as memory, file handles, or hardware connections, and allows you to change properties and input characteristics of `obj`.

Input Arguments

`obj` — Object to release
object

Object you want to release.

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

phased.BackscatterRadarTarget

Package: `phased`

Backscatter radar target

Description

The `phased.BackscatterRadarTarget` System object models the backscatter of a signal from a target. Backscattering is a special case of radar target scattering when the incident and reflected angles are the same. This type of scattering applies to monostatic radar configurations. The radar cross-section determines the backscattering response of a target to an incident signal. This System object lets you specify an angle-dependent radar cross-section model that covers a range of incident angles.

The `phased.BackscatterRadarTarget` System object creates a backscattered signal for polarized and nonpolarized signals. While electromagnetic radar signals are polarized, you can often ignore polarization in your simulation and process the signals as scalar signals. To ignore polarization, specify the `EnablePolarization` property as `false`. To employ polarization, specify the `EnablePolarization` property as `true`.

For nonpolarized signals, you specify the radar cross section as an array of radar cross-section (RCS) values at discrete azimuth and elevation points. The System object interpolates values for incident angles between array points. For polarized signals, you specify the *radar scattering matrix* using three arrays defined at discrete azimuth and elevation points. These three arrays correspond to the *HH*, *HV*, and *VV* polarization components. The *VH* component is computed using the conjugate symmetry property of the *HV* component.

For both nonpolarized and polarized signal cases, you can employ one of four Swerling models to generate random fluctuations in the RCS or radar scattering matrix. Choose the model using the `Model` property. Then, use the `SeedSource` and `Seed` properties to control the fluctuations.

EnablePolarization	Use these properties
`false`	RCSPattern
`true`	ShhPattern, SvvPattern, and ShvPattern

To perform radar backscattering:

1. Create the `phased.BackscatterRadarTarget` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
target = phased.BackscatterRadarTarget
```

```
target = phased.BackscatterRadarTarget(Name,Value)
```

Description

`target = phased.BackscatterRadarTarget` creates a backscatter radar target System object, `target`.

`target = phased.BackscatterRadarTarget(Name,Value)` creates a backscatter radar target System object, `target`, with each specified property `Name` set to the specified `Value`. You can specify additional name and value pair arguments in any order as (`Name1,Value1`,...,`NameN,ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`EnablePolarization` — Enable polarized signals
`false` (default) | `true`

Option to enable processing of polarized signals, specified as `false` or `true`. Set this property to `true` to allow the target to simulate the reflection of polarized radiation. Set this property to `false` to ignore polarization.

Example: `true`

Data Types: `logical`

`AzimuthAngles` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector | *P*-by-1 real-valued column vector

Azimuth angles used to define the angular coordinates of each column of the matrices specified by the `RCSPattern`, `ShhPattern`, `ShvPattern`, or `SvvPattern` properties. Specify the azimuth angles as a length *P* vector. *P* must be greater than two. Angle units are in degrees.

Example: `[-45:0.1:45]`

Data Types: `double`

`ElevationAngles` — Elevation angles
`[-90:90]` (default) | 1-by-*Q* real-valued row vector | *Q*-by-1 real-valued column vector

Elevation angles used to define the angular coordinates of each row of the matrices specified by the `RCSPattern`, `ShhPattern`, `ShvPattern`, or `SvvPattern` properties. Specify the elevation angles as a length *Q* vector. *Q* must be greater than two. Angle units are in degrees.

Example: `[-30:0.1:30]`

Data Types: `double`

`RCSPattern` — Radar cross-section pattern
`ones(181,361)` (default) | *Q*-by-*P* real-valued matrix | *Q*-by-*P*-by-*M* real-valued array | 1-by-*P* real-valued vector | *M*-by-*P* real-valued matrix

Radar cross-section (RCS) pattern, specified as a *Q*-by-*P* real-valued matrix or a *Q*-by-*P*-by-*M* real-valued array. *Q* is the length of the vector in the `ElevationAngles` property. *P* is the length of the vector in the `AzimuthAngles` property. *M* is the number of RCS patterns. The number of patterns corresponds to the number of signals `sig` passed into the function. You can, however, use a single pattern to model multiple signals reflecting from a single target. Pattern units are square-meters.

You can also specify the pattern as a function only of azimuth for a single elevation. In this case, specify the pattern as either a 1-by-*P* vector or an *M*-by-*P* matrix. Each row is a separate pattern.

This property applies when the `EnablePolarization` property is `false`.

Example: `[1,.5;.5,1]`

Data Types: `double`

`ShhPattern` — Radar-scattering matrix *HH* polarization component
`ones(181,361)` (default) | *Q*-by-*P* complex-valued matrix | *Q*-by-*P*-by-*M* complex-valued array | 1-by-*P* complex-valued vector | *M*-by-*P* complex-valued matrix

Radar scattering matrix *HH* polarization component, specified as a *Q*-by-*P* complex-valued matrix or a *Q*-by-*P*-by-*M* complex-valued array. *Q* is the length of the vector in the `ElevationAngles` property. *P* is the length of the vector in the `AzimuthAngles` property. *M* is the number of target patterns. The number of patterns corresponds to the number of signals `sig` passed into the function. You can, however, use a single pattern to model multiple signals reflecting from a single target. Scattering matrix units are meters.

You can also specify the pattern as a function only of azimuth for a single elevation. Then, specify the pattern as either a 1-by-*P* vector or an *M*-by-*P* matrix. Each row is a separate pattern.

This property applies when the `EnablePolarization` property is `true`.

Example: `[1,1;1i,1i]`

Data Types: `double`
Complex Number Support: Yes

`SvvPattern` — Radar scattering matrix *VV* polarization component
`ones(181,361)` (default) | *Q*-by-*P* complex-valued matrix | *Q*-by-*P*-by-*M* complex-valued array | 1-by-*P* complex-valued vector | *M*-by-*P* complex-valued matrix

Radar scattering matrix *VV* polarization component, specified as a *Q*-by-*P* complex-valued matrix or a *Q*-by-*P*-by-*M* complex-valued array. *Q* is the length of the vector in the `ElevationAngles` property. *P* is the length of the vector in the `AzimuthAngles` property. *M* is the number of target patterns. The number of patterns corresponds to the number of signals `sig` passed into the function. You can, however, use a single pattern to model multiple signals reflecting from a single target. Scattering matrix units are meters.

You can also specify the pattern as a function only of azimuth for a single elevation. In this case, specify the pattern as either a 1-by-*P* vector or an *M*-by-*P* matrix. Each row is a separate pattern.

This property applies when the `EnablePolarization` property is `true`.

Example: `[1,1;1i,1i]`

Data Types: `double`
Complex Number Support: Yes

ShvPattern — Radar scattering matrix *HV* polarization component
ones(181,361) (default) | *Q*-by-*P* complex-valued matrix | *Q*-by-*P*-by-*M* complex-valued array | 1-by-*P* complex-valued vector | *M*-by-*P* complex-valued matrix

Radar scattering matrix *HV* polarization component, specified as a *Q*-by-*P* complex-valued matrix or a *Q*-by-*P*-by-*M* complex-valued array. *Q* is the length of the vector in the ElevationAngles property. *P* is the length of the vector in the AzimuthAngles property. *M* is the number of target patterns. The number of patterns corresponds to the number of signals aig passed into the function. You can, however, use a single pattern to model multiple signals reflecting from a single target. Scattering matrix units are meters.

You can also specify the pattern as a function only of azimuth for a single elevation. In this case, specify the pattern as either a 1-by-*P* vector or an *M*-by-*P* matrix. Each row is a separate pattern.

This property applies when the EnablePolarization property is true.

Example: [1,1;1i,1i]

Data Types: double
Complex Number Support: Yes

Model — Target fluctuation model
'Nonfluctuating' (default) | 'Swerling1' | 'Swerling2' | 'Swerling3' | 'Swerling4'

Target fluctuation model, specified as 'Nonfluctuating', 'Swerling1', 'Swerling2', 'Swerling3', or 'Swerling4'. If you set this property to a value other than 'Nonfluctuating', use the update input argument when calling the function.

Example: 'Swerling3'

Data Types: char

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed'). See physconst for more information.

Example: 3e8

Data Types: double

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: 1e9

Data Types: double

SeedSource — Seed source of random number generator for RCS fluctuation model
'Auto' (default) | 'Property'

Seed source of random number generator for RCS fluctuation model, specified as 'Auto' or 'Property'. When you set this property to 'Auto', the System object generates random numbers using the default MATLAB random number generator. When you set this property to 'Property', you specify the random number generator seed using the `Seed` property. This property applies when you set the `Model` property to`'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. When you use this object with Parallel Computing Toolbox™ software, you set this property to `'Auto'`.

Example: `'Property'`

Data Types: `char`

Seed — Random number generator seed
`0` (default) | nonnegative integer less than 2^{32}

Random number generator seed, specified as a nonnegative integer less than 2^{32}. This property applies when the `SeedSource` property is set to `'Property'`.

Example: `32301`

Data Types: `double`

Usage

Syntax

```
refl_sig = target(sig,ang)
refl_sig = target(sig,ang,update)

refl_sig = target(sig,ang,laxes)
refl_sig = target(sig,ang,laxes,update)
```

Description

`refl_sig = target(sig,ang)` returns the reflected signal, `refl_sig`, of an incident nonpolarized signal, `sig`, arriving at the target from the angle, `ang`. This syntax applies when you set the `EnablePolarization` property to `false` and the `Model` property to `'Nonfluctuating'`. In this case, the values specified in the `RCSPattern` property are used to compute the RCS values for the incident and reflected directions, `ang`.

`refl_sig = target(sig,ang,update)` uses `update` to control whether to update the RCS values. This syntax applies when you set the `EnablePolarization` property to `false` and the `Model` property to one of the fluctuating RCS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new RCS value is generated. If `update` is `false`, the previous RCS value is used.

`refl_sig = target(sig,ang,laxes)` returns the reflected signal, `refl_sig`, of an incident polarized signal, `sig`. The matrix, `laxes`, specifies the local target coordinate system. This syntax applies when you set `EnablePolarization` to `true` and the `Model` property to `'Nonfluctuating'`. The values specified in the `ShhPattern`, `SvvPattern`, and `ShvPattern` properties are used to compute the scattering matrices for the incident and reflected directions, `ang`.

`refl_sig = target(sig,ang,laxes,update)` uses the `update` argument to control whether to update the scattering matrix values. This syntax applies when you set the `EnablePolarization` property to `true` and the `Model` property to one of the fluctuating RCS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new RCS value is generated. If `update` is `false`, the previous RCS value is used.

Input Arguments

sig — Narrowband signal
N-by-*M* complex-valued matrix | 1-by-*M* `struct` array containing complex-valued fields

- Narrowband nonpolarized signal, specified as an *N*-by-*M* complex-valued matrix. The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to an independent signal incident at a different reflecting angle.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
- Narrowband polarized signal, specified as a 1-by-*M* `struct` array containing complex-valued fields. Each `struct` element contains three *N*-by-1 column vectors of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing the polarized signal that reflects from the target.

  For polarized fields, the `struct` element contains three *N*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

ang — Incident signal direction
2-by-1 positive real-valued column vector | 2-by-*M* positive real-valued column matrix

Incident signal direction, specified as a 2-by-1 positive real-valued column vector or a 2-by-*M* positive real-valued column matrix. Each column of `ang` specifies the incident direction of the corresponding signal in the form of an `[AzimuthAngle;ElevationAngle]` pair. Units are degrees. The number of columns in `ang` must match the number of independent signals in `sig`.

Example: `[30;45]`

Data Types: `double`

update — Update RCS
`false` (default) | `true`

Allow the RCS values for fluctuation models to update, specified as `false` or `true`. When `update` is `true`, a new RCS value is generated with each call to the function. If `update` is `false`, the RCS remains unchanged with each call to function.

Example: `true`

Data Types: `logical`

laxes — Local coordinate matrix
`eye(3,3)` (default) | 3-by-3 real-valued orthonormal matrix | 3-by-3-by-*M* real-valued array

Local coordinate system matrix, specified as a 3-by-3 real-valued orthonormal matrix or a 3-by-3-by-*M* real-valued array. The matrix columns specify the local coordinate system orthonormal *x*-axis, *y*-axis, and *z*-axis, respectively. Each axis is a vector of the form *(x;y;z)* with respect to the global coordinate system. When `sig` has only one signal, `laxes` is a 3-by-3 matrix. When `sig` has multiple signals, you can use a single 3-by-3 matrix for multiple signals in `sig`. In this case, all targets have the same local coordinate systems. When you specify `laxes` as a 3-by-3-by-*M* MATLAB array, each page (third index) defines a 3-by-3 local coordinate matrix for the corresponding target.

Example: `[1,0,0;0,0.7071,-0.7071;0,0.7071,0.7071]`

Data Types: `double`

Output Arguments

`refl_sig` — Narrowband reflected signal
N-by-*M* complex-valued matrix | *1*-by-*M* `struct` array containing complex-valued fields

- Narrowband nonpolarized signal, specified as an *N*-by-*M* complex-valued matrix. Each column contains an independent signal reflected from the target.

  The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to a reflecting angle.
- Narrowband polarized signal, specified as a 1-by-*M* `struct` array containing complex-valued fields. Each `struct` element contains three *N*-by-1 column vectors of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing the polarized signal that reflects from the target.

  For polarized fields, the `struct` element contains three *N*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

The output `refl_sig` contains signal samples arriving at the signal destination within the current input time frame. When the propagation time from source to destination exceeds the current time frame duration, the output does not contain all contributions from the input of the current time frame. The remaining output appears in the next call to function.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

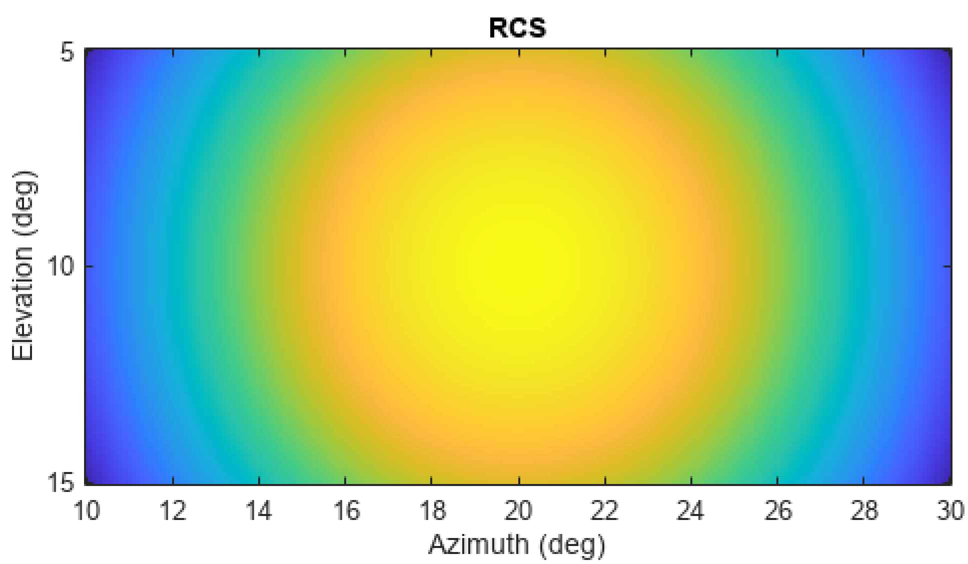
Examples

Backscatter Nonpolarized Signal

Calculate the reflected radar signal from a nonfluctuating point target with a peak RCS of 10.0 m^2. Use a simplified expression of an RCS pattern of a target for illustrative purposes. Real RCS patterns are more complicated. The RCS pattern covers a range of angles from 10°–30° in azimuth and 5°–15° in elevation. The RCS peaks at 20° azimuth and 10° elevation. Assume that the radar operating frequency is 1 GHz and that the signal is a sinusoid at 1 MHz.

Create and plot the RCS pattern.

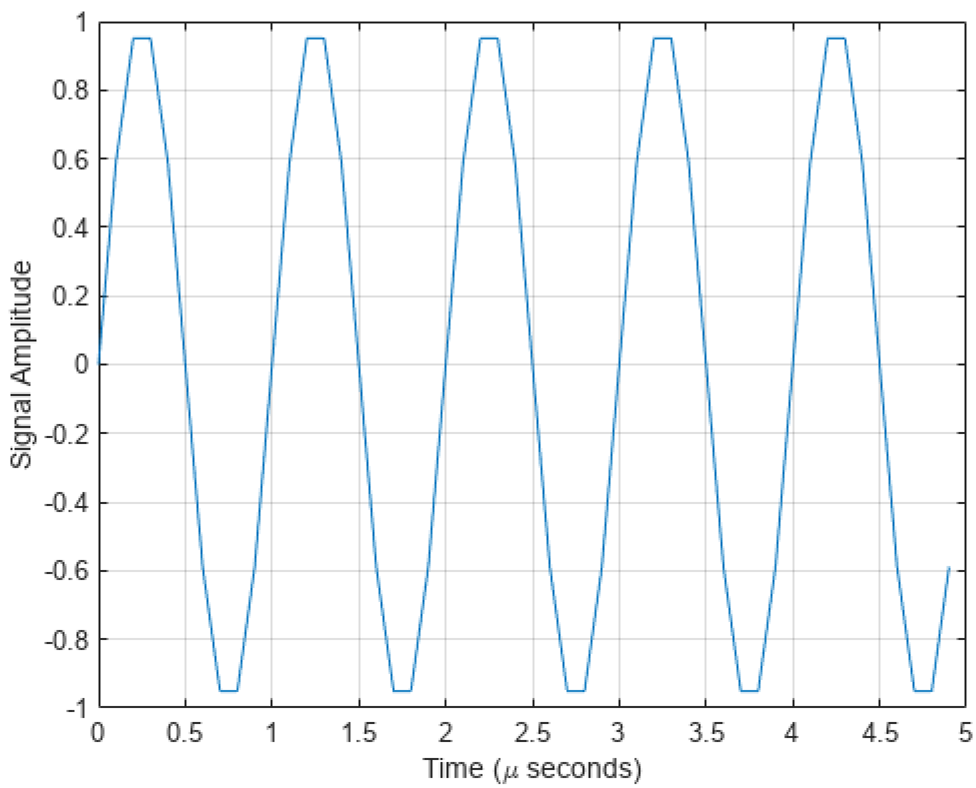
```
azmax = 20.0;
elmax = 10.0;
azpatangs = 10.0:0.1:30.0;
elpatangs = 5.0:0.1:15.0;
rcspattern = 10.0*cosd(4*(elpatangs - elmax))'*cosd(4*(azpatangs - azmax));
imagesc(azpatangs,elpatangs,rcspattern)
axis image
axis tight
title('RCS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
```



Generate and plot 50 samples of the radar signal.

```
foper = 1.0e9;
freq = 1.0e6;
fs = 10*freq;
```

```
nsamp = 50;
t = (0:(nsamp-1))'/fs;
sig = sin(2*pi*freq*t);
plot(t*1e6,sig)
xlabel('Time (\mu seconds)')
ylabel('Signal Amplitude')
grid
```



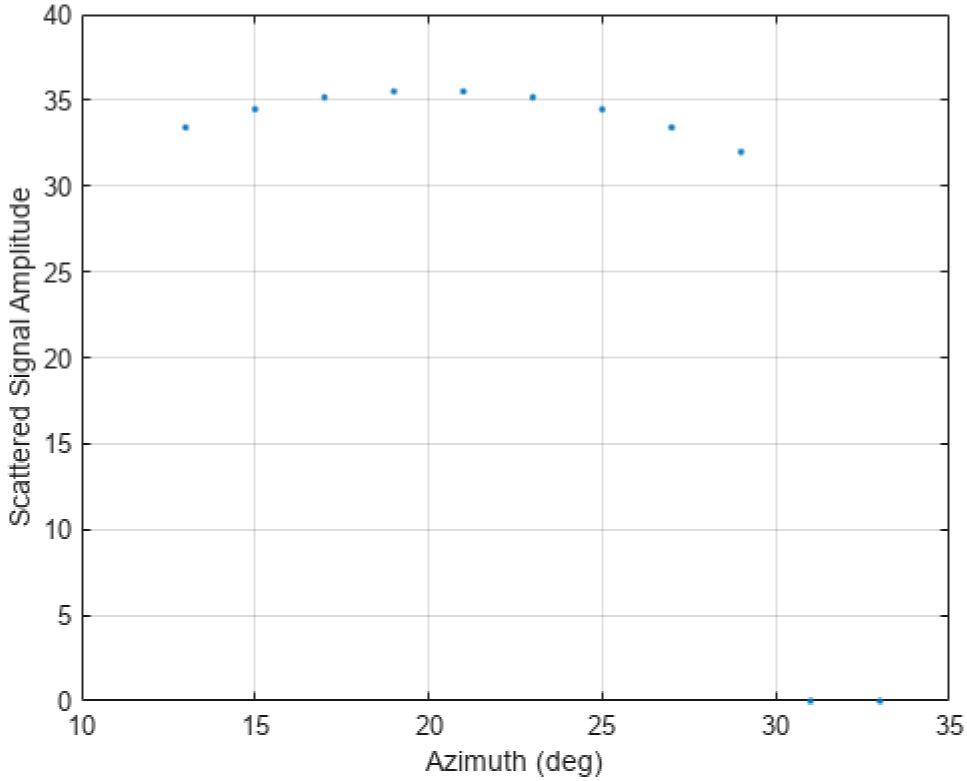
Create the `phased.BackscatterRadarTarget` System object™.

```
target = phased.BackscatterRadarTarget('Model','Nonfluctuating',...
    'AzimuthAngles',azpatangs,'ElevationAngles',elpatangs,...
    'RCSPattern',rcspattern,'OperatingFrequency',foper);
```

For a sequence of incident angles at constant elevation angle, find and plot the scattered signal amplitude.

```
az0 = 13.0;
el = 10.0;
az = az0 + (0:2:20);
naz = length(az);
ss = zeros(1,naz);
for k = 1:naz
    y = target(sig,[az(k);el]);
    ss(k) = max(abs(y));
end
plot(az,ss,'.')
xlabel('Azimuth (deg)')
```

```
ylabel('Scattered Signal Amplitude')
grid
```



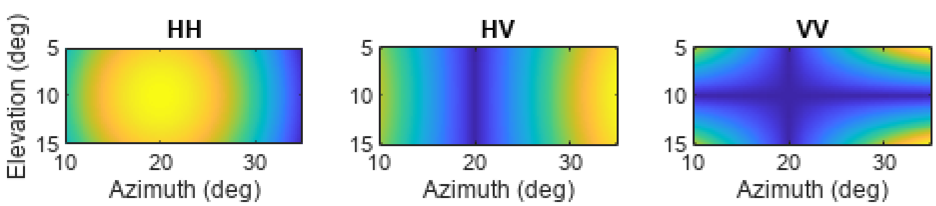
Backscatter Polarized Signal

Calculate the polarized radar signal scattered from a Swerling1 fluctuating point target. Assume the target axis is rotated from the global coordinate system. Use simple expressions for the scattering patterns for illustration. Real scattering patterns are more complicated. For polarized signals, you need to specify the *HH*, *HV*, and *VV* components of the scattering matrix for a range of incident angles. In this example, the patterns cover the range 10°–30° in azimuth and 5°–15° in elevation. Angles are with respect to the target local coordinate system. Assume that the radar operating frequency is 1 GHz and that the signal is a sinusoid with a frequency of 1 MHz. The incident angle is 13.0° azimuth and 14.0° elevation with respect to the target orientation.

Create and plot the scattering matrix patterns.

```
azmax = 20.0;
elmax = 10.0;
azpatangs = (10.0:0.1:35.0);
elpatangs = (5.0:0.1:15.0);
shhpat = cosd(4*(elpatangs - elmax))'*cosd(4*(azpatangs - azmax));
shvpat = 1i*cosd(4*(elpatangs - elmax))'*sind(4*(azpatangs - azmax));
svvpat = sind(4*(elpatangs - elmax))'*sind(4*(azpatangs - azmax));
subplot(1,3,1)
```

```
imagesc(azpatangs,elpatangs,abs(shhpat))
axis image
axis tight
title('HH')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
subplot(1,3,2)
imagesc(azpatangs,elpatangs,abs(shvpat))
axis image
axis tight
title('HV')
xlabel('Azimuth (deg)')
subplot(1,3,3)
imagesc(azpatangs,elpatangs,abs(svvpat))
axis image
axis tight
title('VV')
xlabel('Azimuth (deg)')
```



Create the `phased.BackscatterRadarTarget` System object™.

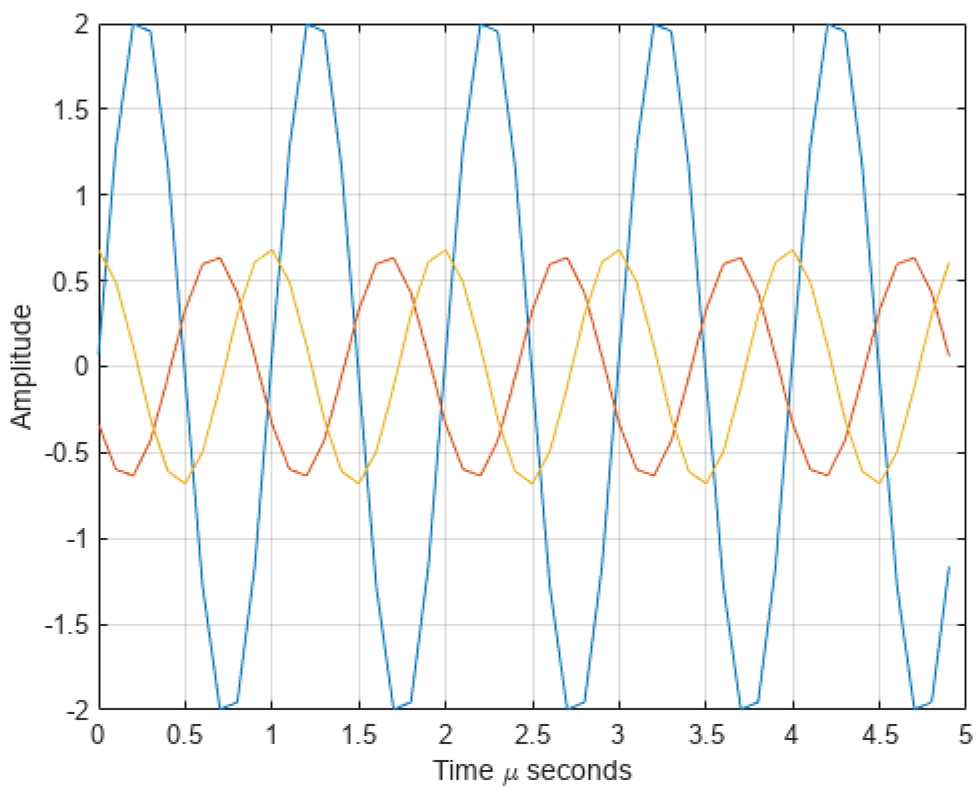
```
target = phased.BackscatterRadarTarget('EnablePolarization',true,...
    'Model','Swerling1','AzimuthAngles',azpatangs,...
    'ElevationAngles',elpatangs,'ShhPattern',shhpat,'ShvPattern',shvpat,...
    'SvvPattern',svvpat);
```

Generate 50 samples of a polarized radar signal.

```
foper = 1.0e9;
freq = 1.0e6;
fs = 10*freq;
nsamp = 50;
t = (0:(nsamp-1))'/fs;
signal.X = exp(1i*2*pi*freq*t);
signal.Y = exp(1i*2*pi*freq*t + pi/3);
signal.Z = zeros(size(signal.X));
tgtaxes = azelaxes(60,10);
ang = [13.0;14.0];
```

Reflect the signal from the target and plot its components.

```
refl_signal = target(signal,ang,tgtaxes,true);
figure
plot(t*1e6,real(refl_signal.X))
hold on
plot(t*1e6,real(refl_signal.Y))
plot(t*1e6,real(refl_signal.Z))
hold off
xlabel('Time \mu seconds')
ylabel('Amplitude')
grid
```



More About

Backscattered Radiation

For a narrowband nonpolarized signal, the reflected signal, Y, is

$$Y = \sqrt{G} \cdot X,$$

where:

- X is the incoming signal.
- G is the target gain factor, a dimensionless quantity given by

$$G = \frac{4\pi\sigma}{\lambda^2}.$$

  - σ is the mean radar cross-section (RCS) of the target.
  - λ is the wavelength of the incoming signal.

The incident signal on the target is scaled by the square root of the gain factor.

For narrowband polarized waves, the single scalar signal, X, is replaced by a vector signal, (E_H, E_V), with horizontal and vertical components. The scattering matrix, S, replaces the scalar cross-section, σ. Through the scattering matrix, the incident horizontal and vertical polarized signals are converted into the reflected horizontal and vertical polarized signals.

$$\begin{bmatrix} E_H^{(scat)} \\ E_V^{(scat)} \end{bmatrix} = \sqrt{\frac{4\pi}{\lambda^2}} \begin{bmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{bmatrix} \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix} = \sqrt{\frac{4\pi}{\lambda^2}} [S] \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

For further details, see [1] or [2].

Version History

Introduced in R2016a

References

[1] Mott, H. *Antennas for Radar and Communications*. New York: John Wiley & Sons, 1992.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[3] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.RadarTarget` | `phased.WidebandBackscatterRadarTarget` | `phased.BackscatterSonarTarget` | `backscatterPedestrian` | `backscatterBicyclist`

Topics

"Modeling Target Radar Cross Section" (Radar Toolbox)
"Simulating Test Signals for a Radar Receiver"
"Swerling Target Models"

reset

System object: `phased.BackscatterRadarTarget`
Package: `phased`

Reset states of System object

Syntax

`reset(sBSTgt)`

Description

`reset(sBSTgt)` resets the internal state of the `phased.BackscatterRadarTarget` object, `sBSTgt`. This method resets the random number generator state if `SeedSource` is a property of this System object and has the value `'Property'`.

Input Arguments

`sBSTgt` — Backscatter radar target
System object

Backscatter radar target, specified as a System object.

Example: `phased.BackscatterRadarTarget`

Version History

Introduced in R2016a

step

System object: `phased.BackscatterRadarTarget`
Package: `phased`

Backscatter incoming signal

Syntax

```
refl_sig = step(target,sig,ang)
refl_sig = step(target,sig,ang,update)

refl_sig = step(target,sig,ang,laxes)
refl_sig = step(target,sig,ang,laxes,update)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`refl_sig = step(target,sig,ang)` returns the reflected signal, `refl_sig`, of an incident nonpolarized signal, `sig`, arriving at the target from the angle, `ang`. This syntax applies when you set the `EnablePolarization` property to `false` and the `Model` property to `'Nonfluctuating'`. In this case, the values specified in the `RCSPattern` property are used to compute the RCS values for the incident and reflected directions, `ang`.

`refl_sig = step(target,sig,ang,update)` uses `update` to control whether to update the RCS values. This syntax applies when you set the `EnablePolarization` property to `false` and the `Model` property to one of the fluctuating RCS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new RCS value is generated. If `update` is `false`, the previous RCS value is used.

`refl_sig = step(target,sig,ang,laxes)` returns the reflected signal, `refl_sig`, of an incident polarized signal, `sig`. The matrix, `laxes`, specifies the local target coordinate system. This syntax applies when you set `EnablePolarization` to `true` and the `Model` property to `'Nonfluctuating'`. The values specified in the `ShhPattern`, `SvvPattern`, and `ShvPattern` properties are used to compute the scattering matrices for the incident and reflected directions, `ang`.

`refl_sig = step(target,sig,ang,laxes,update)` uses the `update` argument to control whether to update the scattering matrix values. This syntax applies when you set the `EnablePolarization` property to `true` and the `Model` property to one of the fluctuating RCS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new RCS value is generated. If `update` is `false`, the previous RCS value is used.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`target` — Backscatter target
System object

Backscatter target, specified as a System object.

Example: `phased.BackscatterRadarTarget`

`sig` — Narrowband signal
N-by-*M* complex-valued matrix | *1*-by-*M* `struct` array containing complex-valued fields

- Narrowband nonpolarized signal, specified as an *N*-by-*M* complex-valued matrix. The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to an independent signal incident at a different reflecting angle.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
- Narrowband polarized signal, specified as a 1-by-*M* `struct` array containing complex-valued fields. Each `struct` element contains three *N*-by-1 column vectors of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing the polarized signal that reflects from the target.

  For polarized fields, the `struct` element contains three *N*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

`ang` — Incident signal direction
2-by-*1* positive real-valued column vector | *2*-by-*M* positive real-valued column matrix

Incident signal direction, specified as a *2*-by-*1* positive real-valued column vector or a *2*-by-*M* positive real-valued column matrix. Each column of `ang` specifies the incident direction of the corresponding signal in the form of an `[AzimuthAngle;ElevationAngle]` pair. Units are degrees. The number of columns in `ang` must match the number of independent signals in `sig`.

Example: `[30;45]`

Data Types: `double`

`update` — Update RCS
`false` (default) | `true`

Allow the RCS values for fluctuation models to update, specified as `false` or `true`. When `update` is `true`, a new RCS value is generated with each call to the `step` method. If `update` is `false`, the RCS remains unchanged with each call to `step`.

Example: `true`

Data Types: `logical`

`laxes` — Local coordinate matrix

`eye(3,3)` (default) | 3-by-3 real-valued orthonormal matrix | 3-by-3-by-*M* real-valued array

Local coordinate system matrix, specified as a 3-by-3 real-valued orthonormal matrix or a 3-by-3-by-*M* real-valued array. The matrix columns specify the local coordinate system orthonormal *x*-axis, *y*-axis, and *z*-axis, respectively. Each axis is a vector of the form *(x;y;z)* with respect to the global coordinate system. When `sig` has only one signal, `laxes` is a 3-by-3 matrix. When `sig` has multiple signals, you can use a single 3-by-3 matrix for multiple signals in `sig`. In this case, all targets have the same local coordinate systems. When you specify `laxes` as a 3-by-3-by-*M* MATLAB array, each page (third index) defines a 3-by-3 local coordinate matrix for the corresponding target.

Example: `[1,0,0;0,0.7071,-0.7071;0,0.7071,0.7071]`

Data Types: `double`

Output Arguments

`refl_sig` — Narrowband reflected signal

N-by-*M* complex-valued matrix | *1*-by-*M* `struct` array containing complex-valued fields

- Narrowband nonpolarized signal, specified as an *N*-by-*M* complex-valued matrix. Each column contains an independent signal reflected from the target.

  The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to a reflecting angle.
- Narrowband polarized signal, specified as a 1-by-*M* `struct` array containing complex-valued fields. Each `struct` element contains three *N*-by-1 column vectors of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing the polarized signal that reflects from the target.

  For polarized fields, the `struct` element contains three *N*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

The output `refl_sig` contains signal samples arriving at the signal destination within the current input time frame. When the propagation time from source to destination exceeds the current time frame duration, the output does not contain all contributions from the input of the current time frame. The remaining output appears in the next call to `step`.

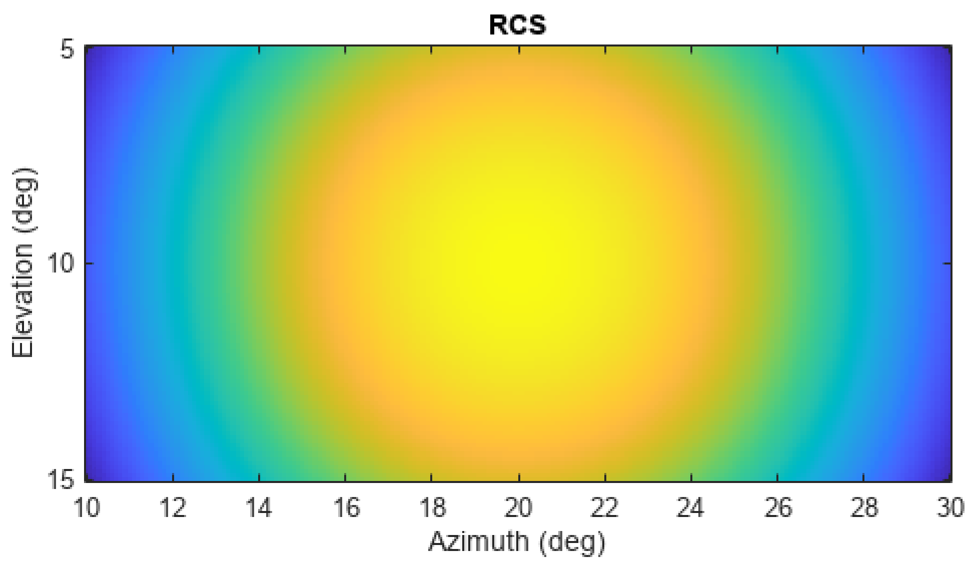
Examples

Backscatter Nonpolarized Signal

Calculate the reflected radar signal from a nonfluctuating point target with a peak RCS of 10.0 m^2. Use a simplified expression of an RCS pattern of a target for illustrative purposes. Real RCS patterns are more complicated. The RCS pattern covers a range of angles from 10°–30° in azimuth and 5°–15° in elevation. The RCS peaks at 20° azimuth and 10° elevation. Assume that the radar operating frequency is 1 GHz and that the signal is a sinusoid at 1 MHz.

Create and plot the RCS pattern.

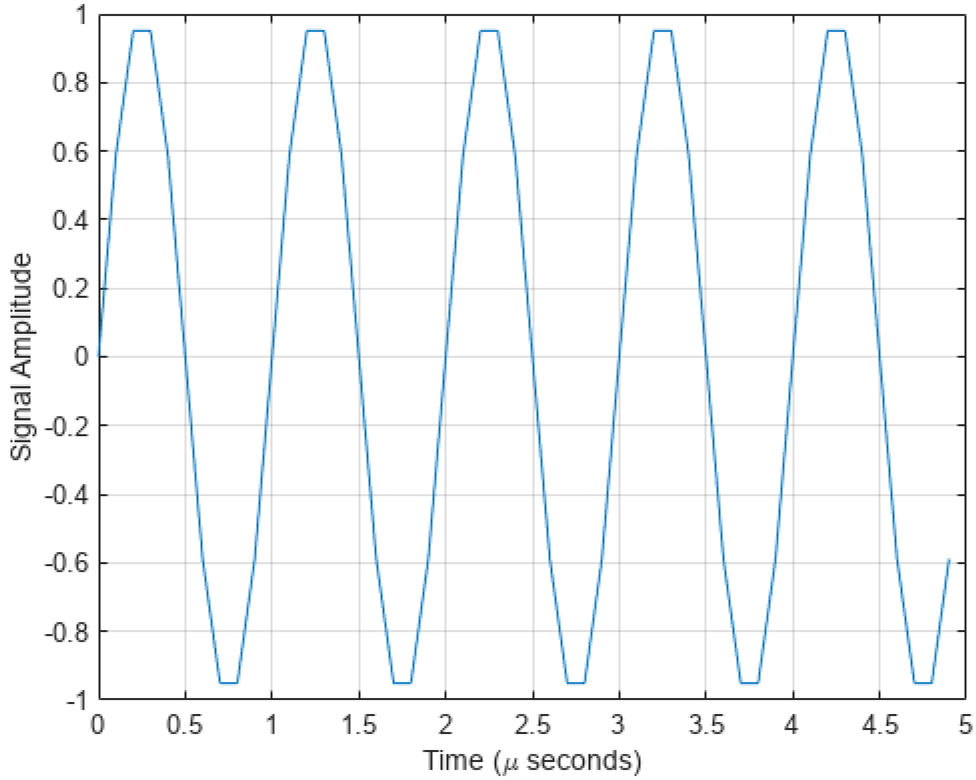
```
azmax = 20.0;
elmax = 10.0;
azpatangs = 10.0:0.1:30.0;
elpatangs = 5.0:0.1:15.0;
rcspattern = 10.0*cosd(4*(elpatangs - elmax))'*cosd(4*(azpatangs - azmax));
imagesc(azpatangs,elpatangs,rcspattern)
axis image
axis tight
title('RCS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
```



Generate and plot 50 samples of the radar signal.

```
foper = 1.0e9;
freq = 1.0e6;
fs = 10*freq;
nsamp = 50;
t = (0:(nsamp-1))'/fs;
sig = sin(2*pi*freq*t);
plot(t*1e6,sig)
xlabel('Time (\mu seconds)')
```

```
ylabel('Signal Amplitude')
grid
```

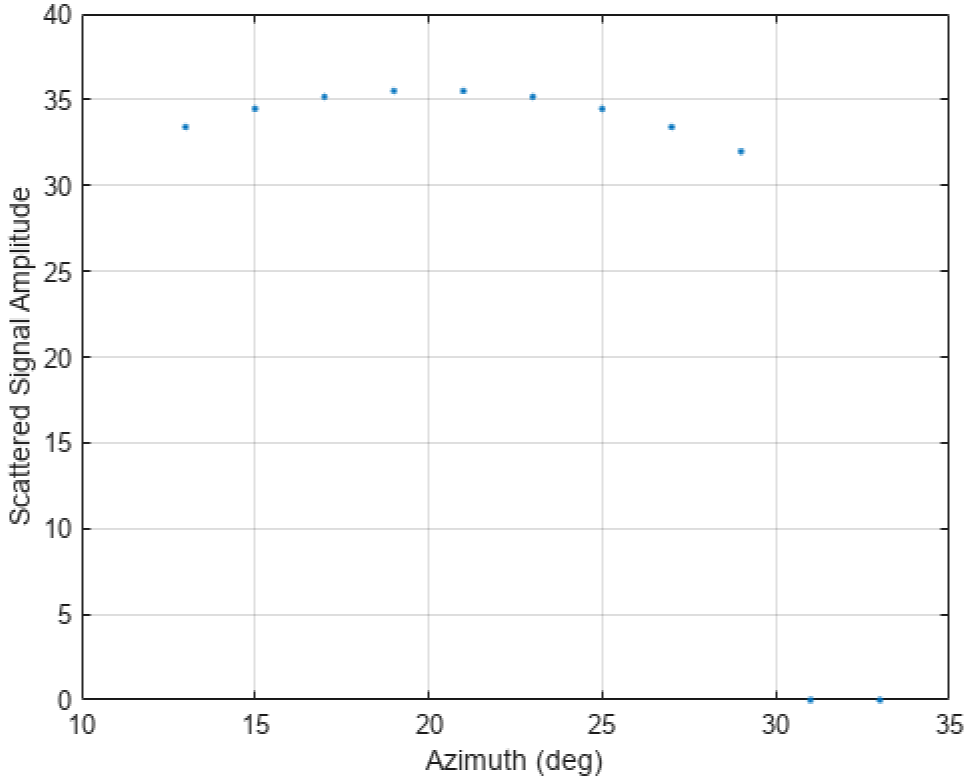


Create the `phased.BackscatterRadarTarget` System object™.

```
target = phased.BackscatterRadarTarget('Model','Nonfluctuating',...
    'AzimuthAngles',azpatangs,'ElevationAngles',elpatangs,...
    'RCSPattern',rcspattern,'OperatingFrequency',foper);
```

For a sequence of incident angles at constant elevation angle, find and plot the scattered signal amplitude.

```
az0 = 13.0;
el = 10.0;
az = az0 + (0:2:20);
naz = length(az);
ss = zeros(1,naz);
for k = 1:naz
    y = target(sig,[az(k);el]);
    ss(k) = max(abs(y));
end
plot(az,ss,'.')
xlabel('Azimuth (deg)')
ylabel('Scattered Signal Amplitude')
grid
```



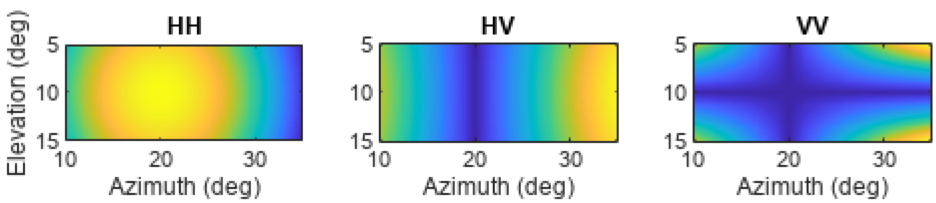
Backscatter Polarized Signal

Calculate the polarized radar signal scattered from a Swerling1 fluctuating point target. Assume the target axis is rotated from the global coordinate system. Use simple expressions for the scattering patterns for illustration. Real scattering patterns are more complicated. For polarized signals, you need to specify the *HH*, *HV*, and *VV* components of the scattering matrix for a range of incident angles. In this example, the patterns cover the range 10°–30° in azimuth and 5°–15° in elevation. Angles are with respect to the target local coordinate system. Assume that the radar operating frequency is 1 GHz and that the signal is a sinusoid with a frequency of 1 MHz. The incident angle is 13.0° azimuth and 14.0° elevation with respect to the target orientation.

Create and plot the scattering matrix patterns.

```
azmax = 20.0;
elmax = 10.0;
azpatangs = (10.0:0.1:35.0);
elpatangs = (5.0:0.1:15.0);
shhpat = cosd(4*(elpatangs - elmax))'*cosd(4*(azpatangs - azmax));
shvpat = 1i*cosd(4*(elpatangs - elmax))'*sind(4*(azpatangs - azmax));
svvpat = sind(4*(elpatangs - elmax))'*sind(4*(azpatangs - azmax));
subplot(1,3,1)
imagesc(azpatangs,elpatangs,abs(shhpat))
axis image
axis tight
```

```
title('HH')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
subplot(1,3,2)
imagesc(azpatangs,elpatangs,abs(shvpat))
axis image
axis tight
title('HV')
xlabel('Azimuth (deg)')
subplot(1,3,3)
imagesc(azpatangs,elpatangs,abs(svvpat))
axis image
axis tight
title('VV')
xlabel('Azimuth (deg)')
```



Create the `phased.BackscatterRadarTarget` System object™.

```
target = phased.BackscatterRadarTarget('EnablePolarization',true,...
    'Model','Swerling1','AzimuthAngles',azpatangs,...
    'ElevationAngles',elpatangs,'ShhPattern',shhpat,'ShvPattern',shvpat,...
    'SvvPattern',svvpat);
```

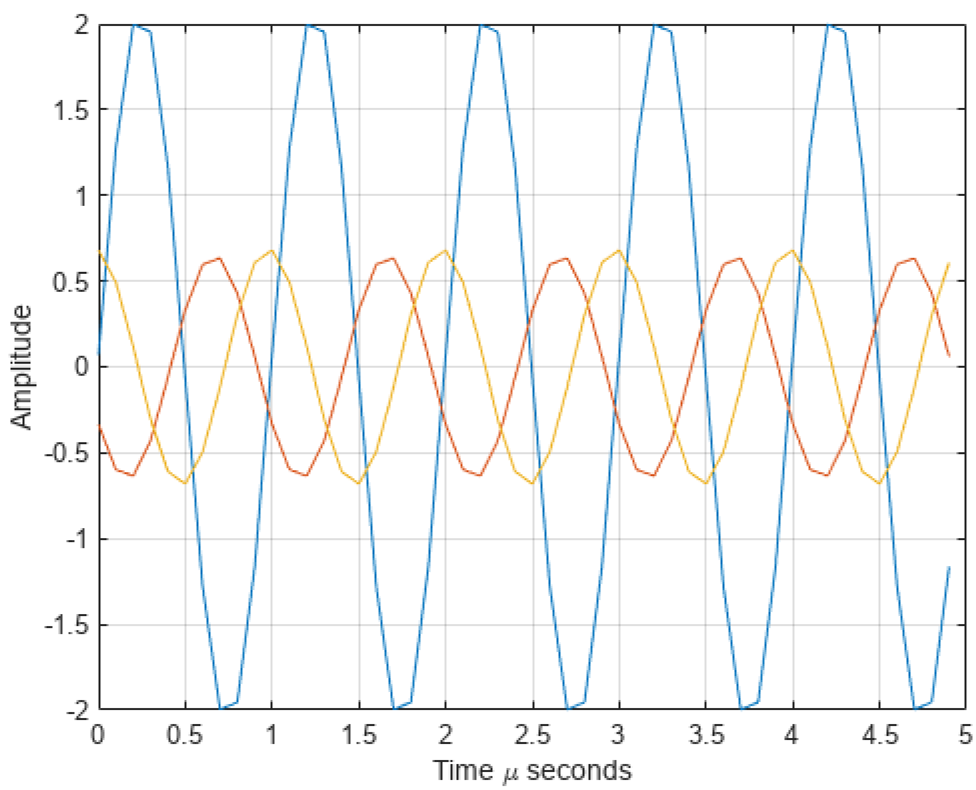
Generate 50 samples of a polarized radar signal.

```
foper = 1.0e9;
freq = 1.0e6;
fs = 10*freq;
```

```
nsamp = 50;
t = (0:(nsamp-1))'/fs;
signal.X = exp(1i*2*pi*freq*t);
signal.Y = exp(1i*2*pi*freq*t + pi/3);
signal.Z = zeros(size(signal.X));
tgtaxes = azelaxes(60,10);
ang = [13.0;14.0];
```

Reflect the signal from the target and plot its components.

```
refl_signal = target(signal,ang,tgtaxes,true);
figure
plot(t*1e6,real(refl_signal.X))
hold on
plot(t*1e6,real(refl_signal.Y))
plot(t*1e6,real(refl_signal.Z))
hold off
xlabel('Time \mu seconds')
ylabel('Amplitude')
grid
```



Version History

Introduced in R2016a

See Also

phased.RadarTarget | phased.WidebandBackscatterRadarTarget

phased.BackscatterSonarTarget

Package: `phased`

Sonar target backscatter

Description

The `phased.BackscatterSonarTarget` System object models the backscattering of a signal from an underwater or surface target. Backscattering is a special case of sonar target scattering when the incident and reflected angles are the same. This type of scattering applies to monostatic sonar configurations. The sonar target strength (TS) determines the backscattering response of a target to an incoming signal. This object lets you specify an angle-dependent sonar target strength model that covers a range of incident angles.

The object lets you specify the target strength as an array of values at discrete azimuth and elevation points. The object interpolates values for incident angles between array points.

You can employ one of four Swerling models to generate random fluctuations in the target strength. Choose the fluctuation model using the `Model` property. Then, use the `SeedSource` and `Seed` properties to control the fluctuations.

To model a backscattered reflected sonar signal:

1. Define and set up your sonar target. You can set `phased.BackscatterSonarTarget` System object properties at construction time or leave them to their default values. See “Construction” on page 1-64. Some properties that you set at construction time can be changed later. These properties are *tunable*.
2. To compute the reflected signal, call the `step` method of `phased.BackscatterSonarTarget`. The output of the method depends on the properties of the `phased.BackscatterSonarTarget` System object. You can change tunable properties at any time.

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`target = phased.BackscatterSonarTarget` creates a backscatter sonar target System object, `target`.

`target = phased.BackscatterSonarTarget(Name,Value)` creates a backscatter sonar target System object, `target`, with each specified property `Name` set to the specified `Value`. You can specify additional name and value pair arguments in any order as (`Name1,Value1`,...,`NameN,ValueN`).

Properties

`AzimuthAngles` — Target strength azimuth angles
`[-180:180]` (default) | real-valued 1-by-*P* row vector | real-valued *P*-by-1 column vector

Target strength azimuth angles, specified as a real-valued 1-by-*P* row vector or *P*-by-1 column vector. These angles define the azimuth coordinates of each column of the matrix specified by the `TSPattern` property. *P* must be greater than two. Angle units are in degrees.

Example: `[-45:0.1:45]`

Data Types: `double`

`ElevationAngles` — Elevation angles
`[-90:90]` (default) | real-valued 1-by-*Q* row vector | real-valued *Q*-by-1 column vector

Target strength elevation angles, specified as a real-valued 1-by-*Q* row vector or *Q*-by-1 column vector. These angles define the elevation coordinates of each row of the matrix specified by the `TSPattern` property. *Q* must be greater than two. Angle units are in degrees.

Example: `[-30:0.1:30]`

Data Types: `double`

`TSPattern` — Sonar target strength pattern
`zeros(181,361)` (default) | *Q*-by-*P* real-valued matrix | *Q*-by-*P*-by-*M* real-valued array | 1-by-*P* real-valued vector | *M*-by-*P* real-valued matrix

Sonar target strength (TS) pattern, specified as a real-valued *Q*-by-*P* matrix or *Q*-by-*P*-by-*M* array. *Q* is the length of the vector in the `ElevationAngles` property. *P* is the length of the vector in the `AzimuthAngles` property. *M* is the number of target patterns. The number of patterns corresponds to the number of signals passed into the `step` method. You can, however, use a single pattern to model multiple signals reflecting from a single target. Pattern units are dB.

You can also specify the pattern as a function only of azimuth for a single elevation. In this case, specify the pattern as either a 1-by-*P* vector or an *M*-by-*P* matrix. Each row is a separate pattern.

Example: `[1,2;3,4]`

Data Types: `double`

`Model` — Target fluctuation model
`'Nonfluctuating'` (default) | `'Swerling1'` | `'Swerling2'` | `'Swerling3'` | `'Swerling4'`

Target fluctuation model, specified as `'Nonfluctuating'`, `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If you set this property to a value other than `'Nonfluctuating'`, use the `update` input argument when calling the `step` method.

Example: `'Swerling3'`

Data Types: `char`

`SeedSource` — Seed source of random number generator for TS fluctuation model
`'Auto'` (default) | `'Property'`

Seed source of random number generator for TS fluctuation model, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the System object generates random numbers using the default MATLAB random number generator. When you set this property to `'Property'`, you specify the random number generator seed using the `Seed` property. This property applies when you set the `Model` property to`'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4`'. When you use this object with Parallel Computing Toolbox software, you set this property to `'Auto'`.

Example: `'Property'`

Data Types: `char`

Seed — Random number generator seed
`0` (default) | nonnegative integer less than 2^{32}

Random number generator seed, specified as a nonnegative integer less than 2^{32}.

Example: `32301`

Dependencies

To enable this property, set the `SeedSource` property to `'Property'`.

Data Types: `double`

Methods

reset	Reset states of System object
step	Backscatter incoming sonar signal

Common to All System Objects	
`release`	Allow System object property value changes

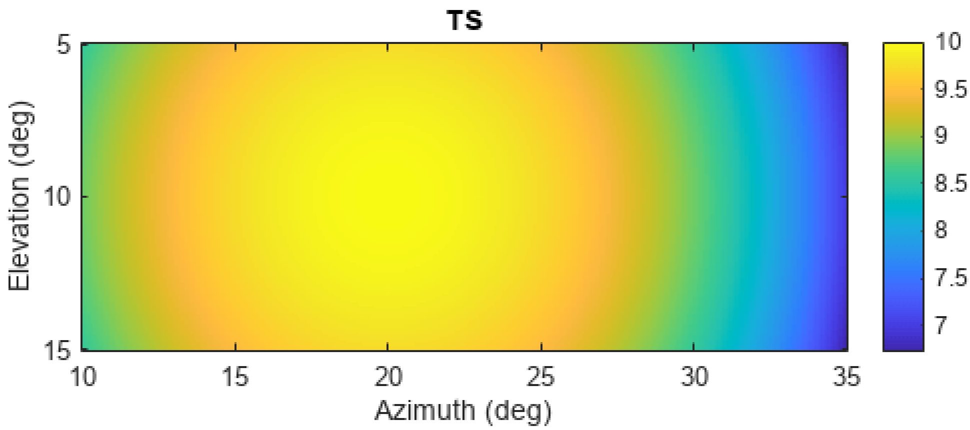
Examples

Backscatter Sonar Signal from Nonfluctuating Target

Calculate the reflected sonar signal from a nonfluctuating point target with a peak target strength (TS) of 10.0 db. For illustrative purposes, use a simplified expression for the TS pattern of a target. Real TS patterns are more complicated. The TS pattern covers a range of angles from 10° to 30° in azimuth and from 5° to 15° in elevation. The TS peaks at 20° azimuth and 10° elevation. Assume that the sonar operating frequency is 10 kHz and that the signal is a sinusoid at 9500 kHz.

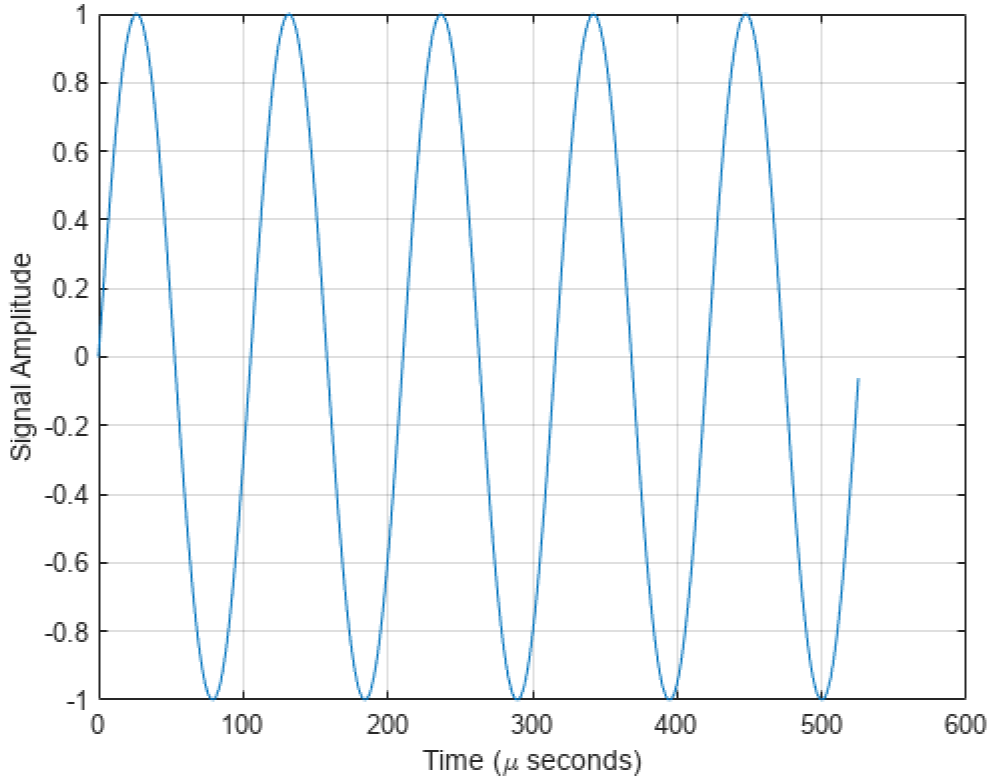
Create and plot the TS pattern.

```
azmax = 20.0;
elmax = 10.0;
azpatangs = [10.0:0.1:35.0];
elpatangs = [5.0:0.1:15.0];
tspattern = 10.0*cosd(4*(elpatangs - elmax))'*cosd(4*(azpatangs - azmax));
tspatterndb = 10*log10(tspattern);
imagesc(azpatangs,elpatangs,tspatterndb)
colorbar
axis image
axis tight
title('TS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
```



Generate and plot 50 samples of the sonar signal.

```
freq = 9.5e3;
fs = 100*freq;
nsamp = 500;
t = [0:(nsamp-1)]'/fs;
sig = sin(2*pi*freq*t);
plot(t*1e6,sig)
xlabel('Time (\mu seconds)')
ylabel('Signal Amplitude')
grid
```

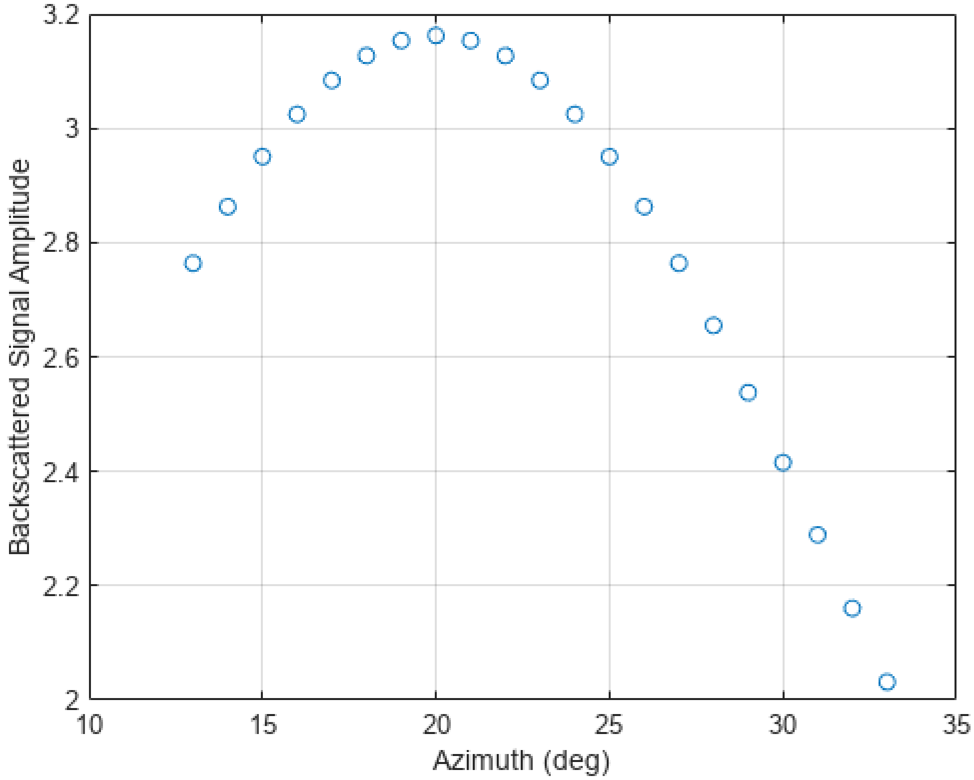


Create the `phased.BackscatterSonarTarget` System object™.

```
target = phased.BackscatterSonarTarget('Model','Nonfluctuating', ...
    'AzimuthAngles',azpatangs,'ElevationAngles',elpatangs, ...
    'TSPattern',tspattern);
```

For a sequence of different azimuth incident angles (at constant elevation angle), plot the maximum scattered signal amplitude.

```
az0 = 13.0;
el = 10.0;
naz = 20;
az = az0 + [0:1:20];
naz = length(az);
ss = zeros(1,naz);
for k = 1:naz
    y = target(sig,[az(k);el]);
    ss(k) = max(abs(y));
end
plot(az,ss,'o')
xlabel('Azimuth (deg)')
ylabel('Backscattered Signal Amplitude')
grid
```

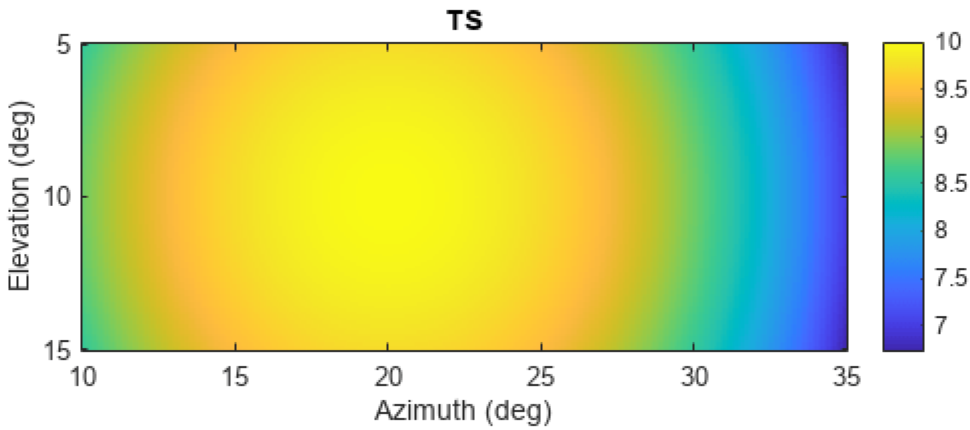


Backscatter Sonar Signal from Fluctuating Target

Calculate the reflected sonar signal from a Swerling2 fluctuating point target with a peak target strength (TS) of 10.0 db. For illustrative purposes, use a simplified expression for the TS pattern of a target. Real TS patterns are more complicated. The TS pattern covers a range of angles from 10°to 30° in azimuth and from 5° ro 15° in elevation. The TS peaks at 20° azimuth and 10° elevation. Assume that the sonar operating frequency is 10 kHz and that the signal is a sinusoid at 9500 kHz.

Create and plot the TS pattern.

```
azmax = 20.0;
elmax = 10.0;
azpatangs = [10.0:0.1:35.0];
elpatangs = [5.0:0.1:15.0];
tspattern = 10.0*cosd(4*(elpatangs - elmax))'*cosd(4*(azpatangs - azmax));
tspatterndb = 10*log10(tspattern);
imagesc(azpatangs,elpatangs,tspatterndb)
colorbar
axis image
axis tight
title('TS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
```



Generate the sonar signal.

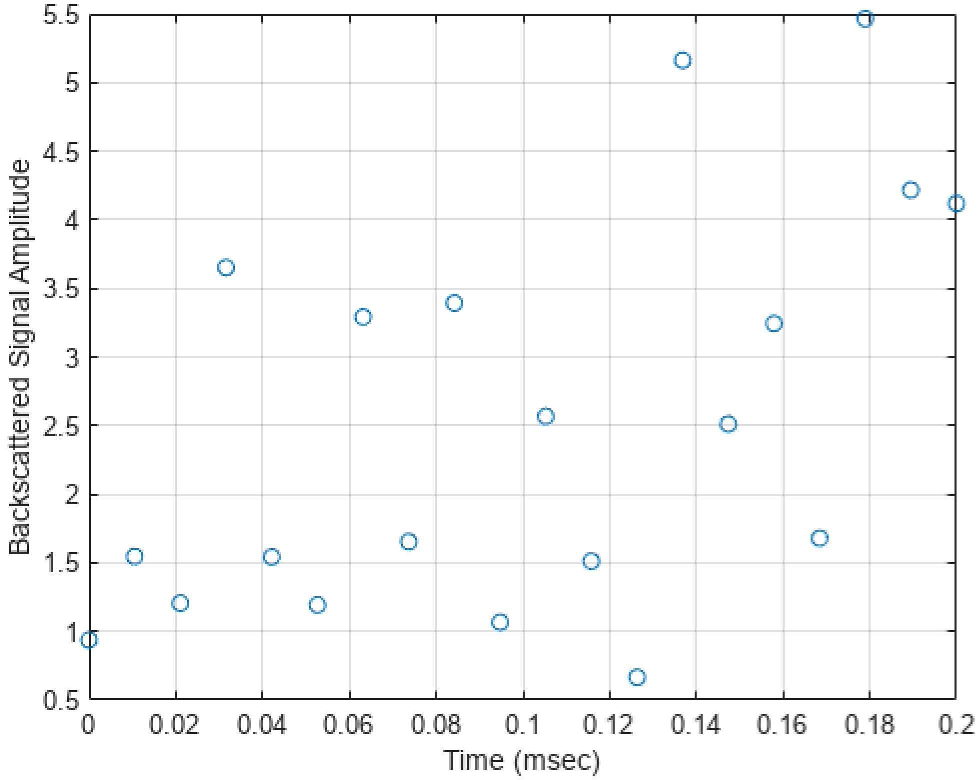
```
freq = 9.5e3;
fs = 10*freq;
nsamp = 50;
t = [0:(nsamp-1)]'/fs;
sig = sin(2*pi*freq*t);
```

Create the `phased.BackscatterSonarTarget` System object™.

```
target = phased.BackscatterSonarTarget('Model','Nonfluctuating',...
    'AzimuthAngles',azpatangs,'ElevationAngles',elpatangs,...
    'TSPattern',tspattern,'Model','Swerling2');
```

Compute and plot the fluctuating signal amplitude for 20 time steps.

```
az = 20.0;
el = 10.0;
nsteps = 20;
ss = zeros(1,nsteps);
for k = 1:nsteps
    y = target(sig,[az;el],true);
    ss(k) = max(abs(y));
end
plot([0:(nsteps-1)]*1000/fs,ss,'o')
xlabel('Time (msec)')
ylabel('Backscattered Signal Amplitude')
grid
```



More About

Backscattered Sound Radiation

For narrowband acoustic signals, the reflected signal, Y, is given by

$$Y = \sqrt{G} \cdot X,$$

where

- X is the incoming signal.
- G is the target gain factor given by $10^{TS/10}$ where TS is the target strength in dB. Specify target strength using the `TSPattern` property.

For a more detailed explanation of target strength, see “[1] [2]” on page 1-71.

Version History

Introduced in R2017a

References

[1] Urick, R.J. *Principles of Underwater Sound, 3rd Edition*. New York: Peninsula Publishing, 1996.

[2] Sherman, C.S., and J. Butler *Transducers and Arrays for Underwater Sound*. New York: Springer, 2007.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects
`phased.BackscatterRadarTarget` | `phased.IsoSpeedUnderwaterPaths` | `phased.WidebandBackscatterRadarTarget` | `phased.RadarTarget` | `backscatterPedestrian` | `backscatterBicyclist`

Topics
"Underwater Target Detection with an Active Sonar System"
"Locating an Acoustic Beacon with a Passive Sonar System"
"Swerling Target Models"

reset

System object: phased.BackscatterSonarTarget
Package: phased

Reset states of System object

Syntax

```
reset(target)
```

Description

`reset(target)` resets the internal state of the `phased.BackscatterSonarTarget` object, `target`. This method resets the random number generator state if `SeedSource` is a property of this System object and has the value `'Property'`.

Input Arguments

`target` — Backscatter sonar target
`phased.BackscatterSonarTarget` System object

Backscatter sonar target, specified as a `phased.BackscatterSonarTarget` System object.

Example: `phased.BackscatterSonarTarget`

Version History

Introduced in R2017a

step

System object: `phased.BackscatterSonarTarget`
Package: `phased`

Backscatter incoming sonar signal

Syntax

```
refl_sig = step(target,sig,ang)
refl_sig = step(target,sig,ang,update)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`refl_sig = step(target,sig,ang)` returns the reflected signal, `refl_sig`, of an incident sonar signal, `sig`, arriving at the target from the angle, `ang`.

`refl_sig = step(target,sig,ang,update)` uses `update` to control whether to update the target strength (TS) values. This syntax applies when you set the `Model` property to one of the fluctuating TS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new TS value is generated. If `update` is `false`, the previous TS value is used.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`target` — Backscatter sonar target

`phased.BackscatterSonarTarget` System object

Backscatter sonar target, specified as a `phased.BackscatterSonarTarget` System object.

`sig` — Sonar signal

N-by-*M* complex-valued matrix

Sonar signal, specified as an *N*-by-*M* complex-valued matrix. The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to an independent signal incident at a different reflecting angle.

When you specify the `TSPattern` property as a *Q*-by-*P*-by-*M*, a separate pattern is used for each signal. When you specify `TSPattern` as a *Q*-by-*P*matrix, the same pattern is used for every signal.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

ang — Incident signal direction
2-by-*1* positive real-valued column vector | *2*-by-*M* positive real-valued column matrix

Incident signal direction, specified as a *2*-by-*1* positive real-valued column vector or a *2*-by-*M* positive real-valued column matrix. Each column of `ang` specifies the incident direction of the corresponding signal in the form of an `[AzimuthAngle;ElevationAngle]` pair. Units are degrees. The number of columns in `ang` must match the number of independent signals in `sig`.

Example: `[30;45]`

Data Types: `double`

`update` — Update target strength
`false` (default) | `true`

Allow the TS values for fluctuation models to update, specified as `false` or `true`. When `update` is `true`, a new TS value is generated with each call to the `step` method. If `update` is `false`, TS remains unchanged with each call to `step`.

Example: `true`

Data Types: `logical`

Output Arguments

`refl_sig` — Narrowband reflected sonar signal
N-by-*M* complex-valued matrix

Narrowband reflected sonar signal, specified as an *N*-by-*M* complex-valued matrix. Each column contains an independent signal reflected from the target.

The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to a reflecting angle.

The output `refl_sig` contains signal samples arriving at the signal destination within the current input time frame. When the propagation time from source to destination exceeds the current time frame duration, the output will not contain all contributions from the input of the current time frame. The remaining output appears in the next call to `step`.

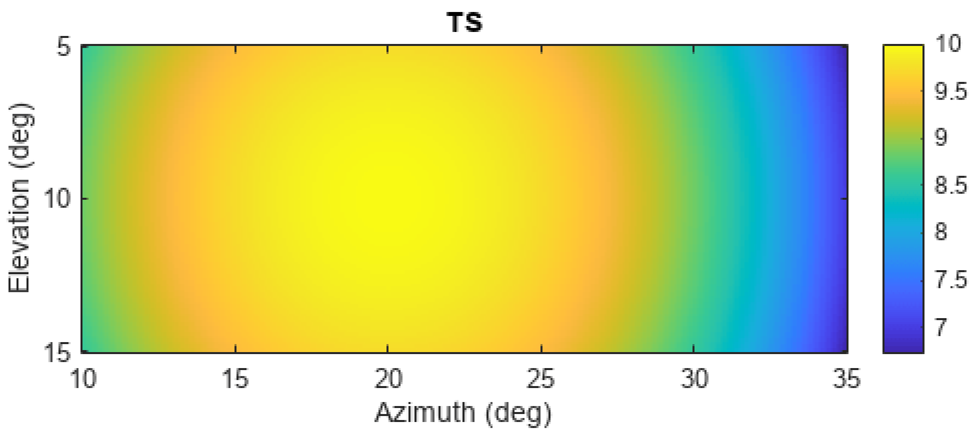
Examples

Backscatter Sonar Signal from Nonfluctuating Target

Calculate the reflected sonar signal from a nonfluctuating point target with a peak target strength (TS) of 10.0 db. For illustrative purposes, use a simplified expression for the TS pattern of a target. Real TS patterns are more complicated. The TS pattern covers a range of angles from 10° to 30° in azimuth and from 5° to 15° in elevation. The TS peaks at 20° azimuth and 10° elevation. Assume that the sonar operating frequency is 10 kHz and that the signal is a sinusoid at 9500 kHz.

Create and plot the TS pattern.

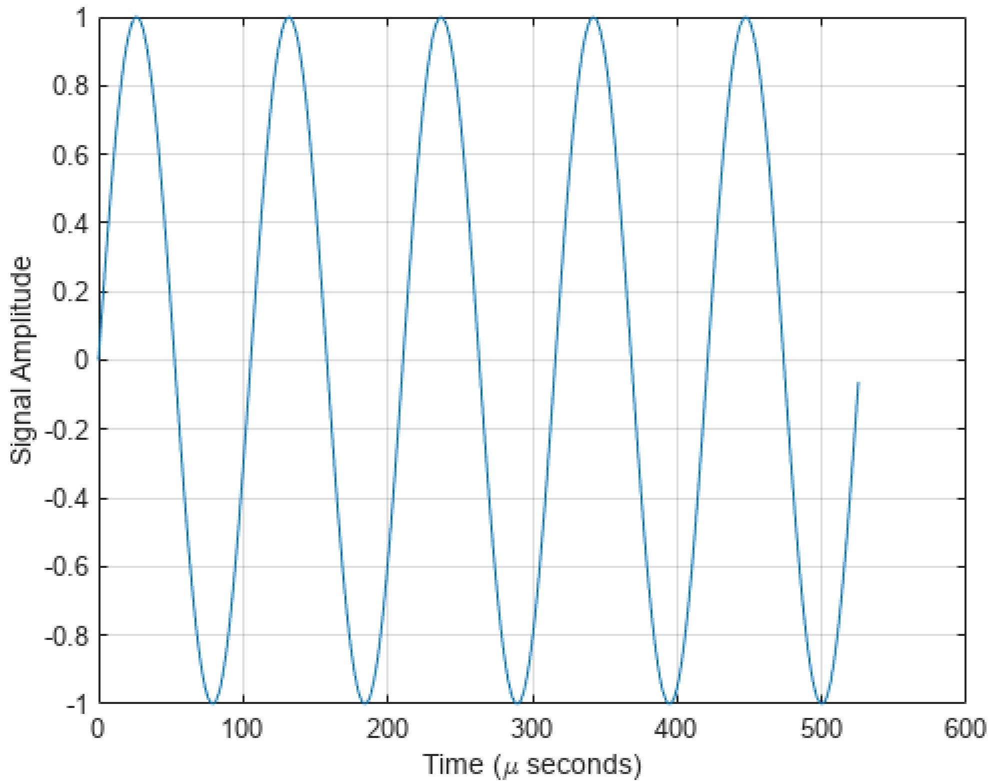
```
azmax = 20.0;
elmax = 10.0;
azpatangs = [10.0:0.1:35.0];
elpatangs = [5.0:0.1:15.0];
tspattern = 10.0*cosd(4*(elpatangs - elmax))'*cosd(4*(azpatangs - azmax));
tspatterndb = 10*log10(tspattern);
imagesc(azpatangs,elpatangs,tspatterndb)
colorbar
axis image
axis tight
title('TS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
```



Generate and plot 50 samples of the sonar signal.

```
freq = 9.5e3;
fs = 100*freq;
nsamp = 500;
t = [0:(nsamp-1)]'/fs;
sig = sin(2*pi*freq*t);
plot(t*1e6,sig)
xlabel('Time (\mu seconds)')
```

```
ylabel('Signal Amplitude')
grid
```

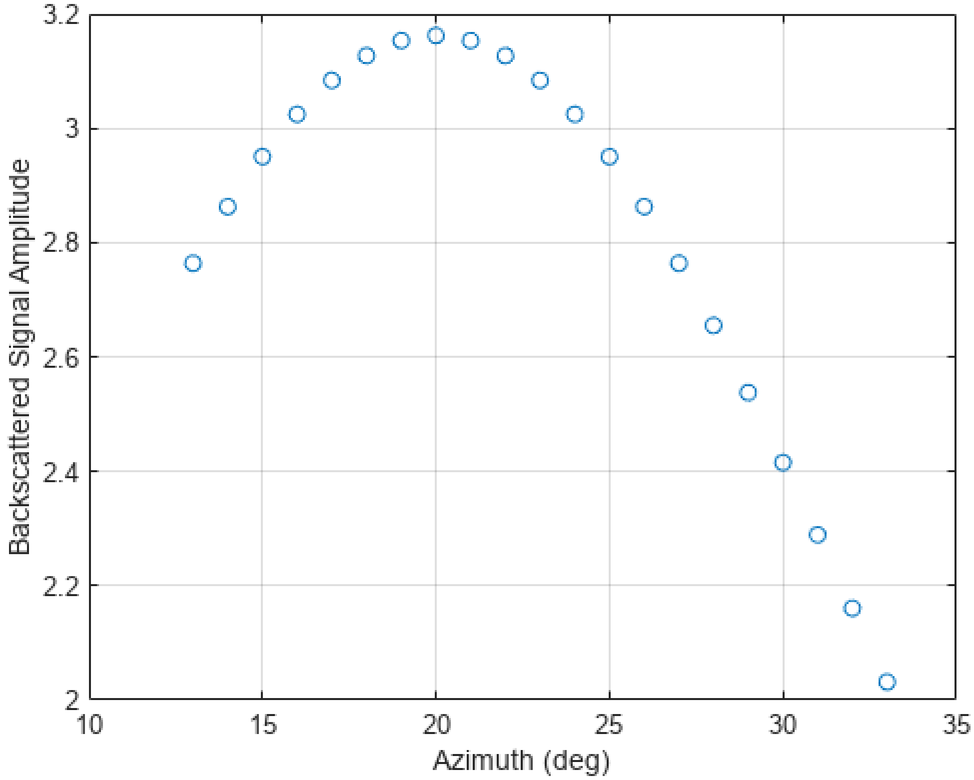


Create the `phased.BackscatterSonarTarget` System object™.

```
target = phased.BackscatterSonarTarget('Model','Nonfluctuating', ...
    'AzimuthAngles',azpatangs,'ElevationAngles',elpatangs, ...
    'TSPattern',tspattern);
```

For a sequence of different azimuth incident angles (at constant elevation angle), plot the maximum scattered signal amplitude.

```
az0 = 13.0;
el = 10.0;
naz = 20;
az = az0 + [0:1:20];
naz = length(az);
ss = zeros(1,naz);
for k = 1:naz
    y = target(sig,[az(k);el]);
    ss(k) = max(abs(y));
end
plot(az,ss,'o')
xlabel('Azimuth (deg)')
ylabel('Backscattered Signal Amplitude')
grid
```

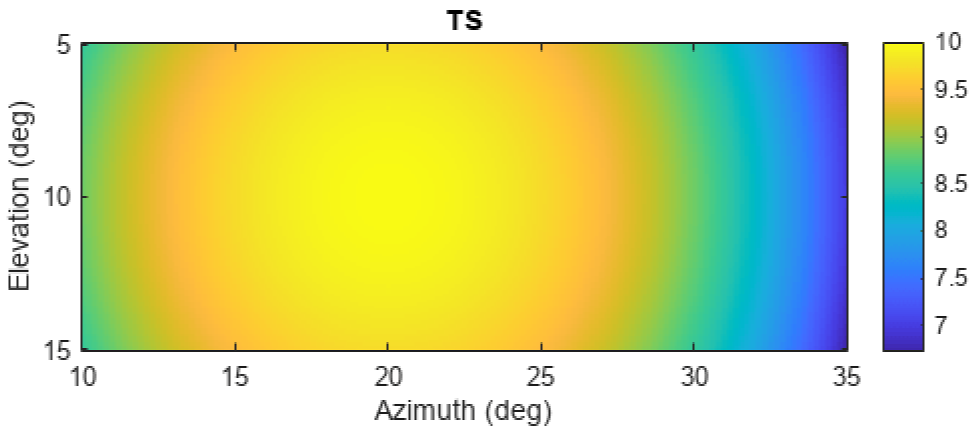


Backscatter Sonar Signal from Fluctuating Target

Calculate the reflected sonar signal from a Swerling2 fluctuating point target with a peak target strength (TS) of 10.0 db. For illustrative purposes, use a simplified expression for the TS pattern of a target. Real TS patterns are more complicated. The TS pattern covers a range of angles from 10°to 30° in azimuth and from 5° ro 15° in elevation. The TS peaks at 20° azimuth and 10° elevation. Assume that the sonar operating frequency is 10 kHz and that the signal is a sinusoid at 9500 kHz.

Create and plot the TS pattern.

```
azmax = 20.0;
elmax = 10.0;
azpatangs = [10.0:0.1:35.0];
elpatangs = [5.0:0.1:15.0];
tspattern = 10.0*cosd(4*(elpatangs - elmax))'*cosd(4*(azpatangs - azmax));
tspatterndb = 10*log10(tspattern);
imagesc(azpatangs,elpatangs,tspatterndb)
colorbar
axis image
axis tight
title('TS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
```



Generate the sonar signal.

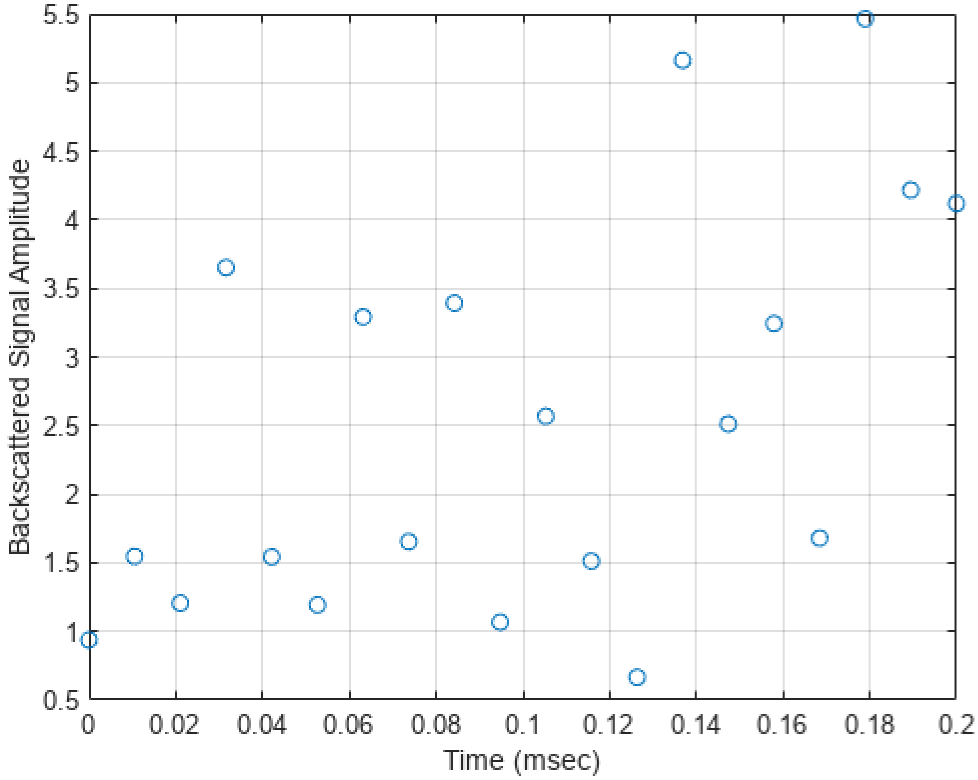
```
freq = 9.5e3;
fs = 10*freq;
nsamp = 50;
t = [0:(nsamp-1)]'/fs;
sig = sin(2*pi*freq*t);
```

Create the `phased.BackscatterSonarTarget` System object™.

```
target = phased.BackscatterSonarTarget('Model','Nonfluctuating',...
    'AzimuthAngles',azpatangs,'ElevationAngles',elpatangs,...
    'TSPattern',tspattern,'Model','Swerling2');
```

Compute and plot the fluctuating signal amplitude for 20 time steps.

```
az = 20.0;
el = 10.0;
nsteps = 20;
ss = zeros(1,nsteps);
for k = 1:nsteps
    y = target(sig,[az;el],true);
    ss(k) = max(abs(y));
end
plot([0:(nsteps-1)]*1000/fs,ss,'o')
xlabel('Time (msec)')
ylabel('Backscattered Signal Amplitude')
grid
```



Version History

Introduced in R2017a

phased.BeamscanEstimator

Package: `phased`

Beamscan spatial spectrum estimator for ULA

Description

The `phased.BeamscanEstimator` System object calculates a beamscan spatial spectrum estimate for a uniform linear array (ULA). The object estimates the incoming signal spatial spectrum using a narrowband conventional beamformer.

To estimate the spatial spectrum:

1. Create the `phased.BeamscanEstimator` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
estimator = phased.BeamscanEstimator
estimator = phased.BeamscanEstimator(Name,Value)
```

Description

`estimator = phased.BeamscanEstimator` creates a beamscan spatial spectrum estimator System object.

`estimator = phased.BeamscanEstimator(Name,Value)` creates an object, `estimator`, with each specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SensorArray` — ULA sensor array
`phased.ULA` System object (default)

ULA sensor array, specified as a `phased.ULA` System object. If you do not specify any name-value pair properties for the ULA sensor array, the default properties of the array are used.

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed').

Example: 3e8

Data Types: single | double

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: 1e9

Data Types: single | double

NumPhaseShifterBits — Number of phase shifter quantization bits
0 (default) | non-negative scalar

The number of bits used to quantize the phase shift component of beamformer or steering vector weights, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Example: 5

Data Types: single | double

ForwardBackwardAveraging — Enable forward-backward averaging
false (default) | true

Enable forward-backward averaging, specified as false or true. Set this property to true to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold.

Data Types: logical

SpatialSmoothing — Enable spatial smoothing
0 (default) | nonnegative integer

Option to enable spatial smoothing, specified as a nonnegative integer. Use spatial smoothing to compute the arrival directions of coherent signals. A value of zero specifies no spatial smoothing. A positive value represents the number of subarrays used to compute the smoothed (averaged) source covariance matrix. Each increment in this value lets you handle one additional coherent source, but reduces the effective number of array elements by one. The length of the smoothing aperture, L, depends on the array length, M, and the averaging number, K, by $L = M - K + 1$. The maximum value of K is $M - 2$.

Example: 5

Data Types: double

ScanAngles — Broadside scan angles
[-90:90] (default) | real-valued K-length vector

Broadside scan angles, specified as a real-valued vector. Units are in degrees. Broadside angles are between the search direction and the ULA array axis. The angles lie between –90° and 90°, inclusive. Specify the angles in increasing value.

Example: `[-20:20]`

Data Types: `single` | `double`

`DOAOutputPort` — Enable directions of arrival output
`false` (default) | `true`

Option to enable directions-of-arrival (DOA) output, specified as `false` or `true`. To obtain the DOA of signals, set this property to `true`. The DOAs are returned in the second output argument when the object is executed.

Data Types: `logical`

`NumSignals` — Number of arriving signals
`1` (default) | positive integer

Number of arriving signals for DOA estimation, specified as a positive integer.

Example: `3`

Dependencies

To enable this property, set the `DOAOutputPort` property to `true`.

Data Types: `single` | `double`

Usage

Syntax

```
Y = estimator(X)
[Y,ANG] = estimator(X)
```

Description

`Y = estimator(X)` estimates the spatial spectrum from data `X`.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`[Y,ANG] = estimator(X)` returns the directions of arrival, `ANG`, of the signals. To enable this syntax, set the `DOAOutputPort` property to `true`. `ANG` is a row vector of the estimated broadside angles (in degrees). You can specify `ANG` as single or double precision. If the object cannot identify a signal direction, it will return `NaN`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

X — Channel data
complex-valued matrix

Channel data, specified as a complex-valued matrix. Columns of the data matrix correspond to channels.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

Y — Magnitude of estimated spatial spectrum
real-valued 1-by-*L* column vector

Magnitude of the estimated spatial spectrum, returned as a real-valued 1-by-*L* column vector. *L* is the number of scan angles specified by the `ScanAngles` property.

Data Types: `single` | `double`

ANG — Estimated broadside angles
real-valued 1-by-*K* row vector | `NaN`

Estimated broadside angles of signal arrivals, returned as a real-valued 1-by-*K* row vector. Units are in degrees. The `NaN` value in any vector element indicates that an estimate could not be found.

Data Types: `single` | `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to spectral estimation

plotSpectrum	Plot spatial spectrum

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Estimate Directions of Arrival of Two Signals

Estimate the DOA's of two signals received by a 10-element ULA with element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 60° in azimuth and -5° in elevation.

Create the signals and array.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
antenna = phased.IsotropicAntennaElement('FrequencyRange',[100e6 300e6]);
array = phased.ULA('Element',antenna,'NumElements',10,'ElementSpacing',1);
fc = 150e6;
x = collectPlaneWave(array,[x1 x2],[10 20;60 -5]',fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
```

Solve for the DOAs.

```
estimator = phased.BeamscanEstimator('SensorArray',array, ...
    'OperatingFrequency',fc,'DOAOutputPort',true,'NumSignals',2);
[~,doas] = estimator(x + noise);
doas = broadside2az(sort(doas),[20 -5]);
disp(doas)
```

```
    9.5829   60.3813
```

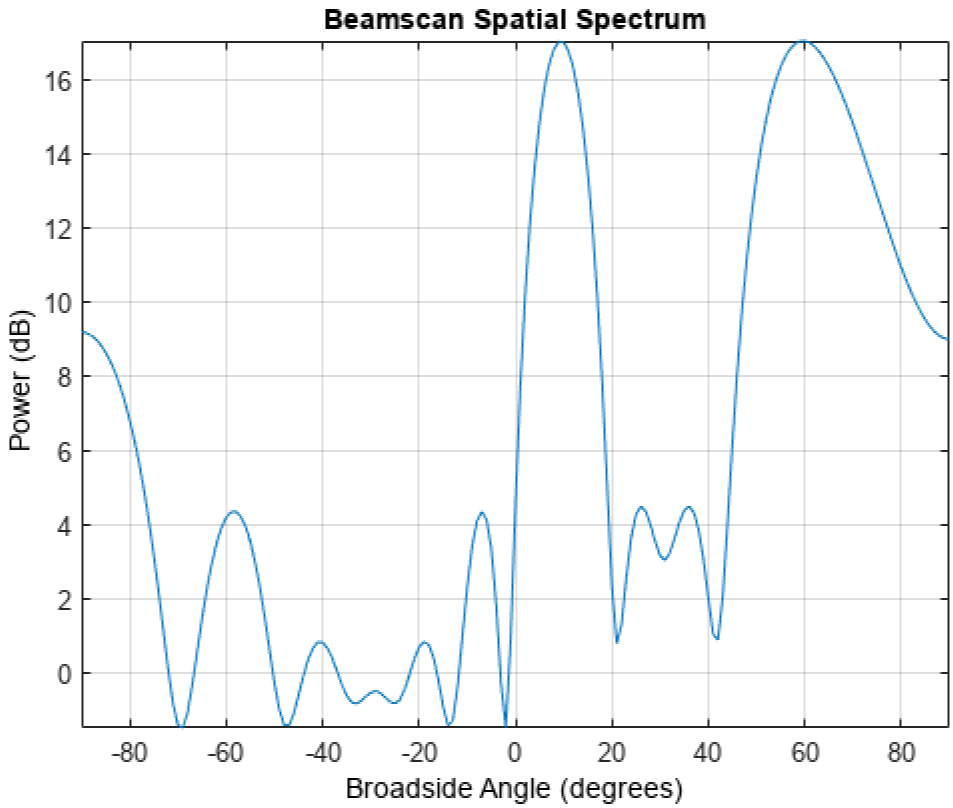
Because the default values for the `ScanAngles` property has a granularity of 1°, the DOA estimates are not accurate. Improve the accuracy by choosing a finer grid.

```
estimator2 = phased.BeamscanEstimator('SensorArray',array, ...
    'OperatingFrequency',fc,'ScanAngles',-60:0.1:60, ...
    'DOAOutputPort',true,'NumSignals',2);
[~,doas] = estimator2(x + noise);
doas = broadside2az(sort(doas),[20 -5]);
disp(doas)
```

```
   10.0093   59.9751
```

Plot the beamscan spectrum

```
plotSpectrum(estimator)
```



Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002, pp. 1142–1143.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`broadside2az` | `phased.BeamscanEstimator2D`

phased.BeamscanEstimator2D

Package: `phased`

2-D beamscan spatial spectrum estimator

Description

The `phased.BeamscanEstimator2D` System object calculates a beamscan 2-D spatial spectrum estimate for any Phased Array System Toolbox array. The object estimates the incoming signal spatial spectrum using a narrowband conventional beamformer.

To estimate the spatial spectrum:

1. Create the `phased.BeamscanEstimator2D` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
estimator = phased.BeamscanEstimator2D
estimator = phased.BeamscanEstimator2D(Name,Value)
```

Description

`estimator = phased.BeamscanEstimator2D` creates a beamscan 2-D spatial spectrum estimator System object.

`estimator = phased.BeamscanEstimator2D(Name,Value)` creates an object, `estimator`, with each specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SensorArray` — Sensor array
phased.ULA array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`.

Example: `3e8`

Data Types: `single` | `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `single` | `double`

`NumPhaseShifterBits` — Number of phase shifter quantization bits
`0` (default) | non-negative scalar

The number of bits used to quantize the phase shift component of beamformer or steering vector weights, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Example: `5`

Data Types: `single` | `double`

`ForwardBackwardAveraging` — Enable forward-backward averaging
`false` (default) | `true`

Enable forward-backward averaging, specified as `false` or `true`. Set this property to `true` to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold.

Data Types: `logical`

`AzimuthScanAngles` — Azimuth scan angles
`[-90:90]` (default) | real-valued row vector

Azimuth scan angles, specified as a or real-valued row vector. Angle units are in degrees. The angle values must lie between –180° and 180°, inclusive, and be in ascending order.

Example: `[-30:20]`

Data Types: `single` | `double`

`ElevationScanAngles` — Elevation scan angles
`0` (default) | real-valued row vector

Elevation scan angles, specified as a real-valued row vector. Angle units are in degrees. The angle values must lie between –90° and 90°, inclusive, and be in ascending order.

Example: `[-70:75]`

Data Types: `single` | `double`

DOAOutputPort — Enable directions of arrival output
false (default) | true

Option to enable directions-of-arrival (DOA) output, specified as false or true. To obtain the DOA of signals, set this property to true. The DOAs are returned in the second output argument when the object is executed.

Data Types: logical

NumSignals — Number of arriving signals
1 (default) | positive integer

Number of arriving signals for DOA estimation, specified as a positive integer.

Example: 3

Dependencies

To enable this property, set the DOAOutputPort property to true.

Data Types: single | double

Usage

Syntax

```
Y = estimator(X)
[Y,ANG] = estimator(X)
```

Description

Y = estimator(X) estimates the spatial spectrum from data X.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

[Y,ANG] = estimator(X) returns the directions of arrival, ANG, of the signals. To enable this syntax, set the DOAOutputPort property to true. ANG is a 2-by-*N* matrix of the estimated azimuths and elevations of the signal direction. *N* is specified by the NumSignals property. If the object cannot identify a signal direction, it will return NaN.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Input Arguments

X — Array data
complex-valued matrix

Array data, specified as a complex-valued matrix. Columns of the data matrix correspond to channels.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

Y — Magnitude of estimated spatial spectrum
positive, real-valued, *K*-by-*L* matrix

Magnitude of the estimated spatial spectrum, returned as a positive, real-valued, *K*-by-*L* matrix.

Data Types: `single` | `double`

ANG — Estimated direction angles of signal arrivals
real-valued 2-by-*K* matrix | `NaN`

Estimated direction angles of signal arrivals, returned as a real-valued 2-by-*K* matrix. Each column has the form `[azimuth;elevation]`. The `NaN` value in any matrix element indicates that an estimate could not be found. Units are in degrees.

Data Types: `single` | `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to spectral estimation

plotSpectrum	Plot spatial spectrum

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Estimate Directions of Arrival of Two Signals

Estimate the DOAs of two signals received by a 50-element URA with a rectangular lattice. The antenna operating frequency is 150 MHz. The actual direction of the first signal is -37° in azimuth and 0° in elevation. The direction of the second signal is 17° in azimuth and 20° in elevation.

```
antenna = phased.IsotropicAntennaElement('FrequencyRange',[100e6 300e6]);
array = phased.URA('Element',antenna,'Size',[5 10],'ElementSpacing',[1 0.6]);
fc = 150e6;
lambda = physconst('LightSpeed')/fc;
ang1 = [-37.5; 10.2];
ang2 = [17.4; 20.6];
x = sensorsig(getElementPosition(array)/lambda,8000,[ang1 ang2],0.2);
```

```
estimator = phased.BeamscanEstimator2D('SensorArray',array,'OperatingFrequency',fc, ...
    'DOAOutputPort',true,'NumSignals',2,'AzimuthScanAngles',-50:50,'ElevationScanAngles',-30:30)
[~,doas] = estimator(x);
disp(doas)
```

```
    17   -37
    20    10
```

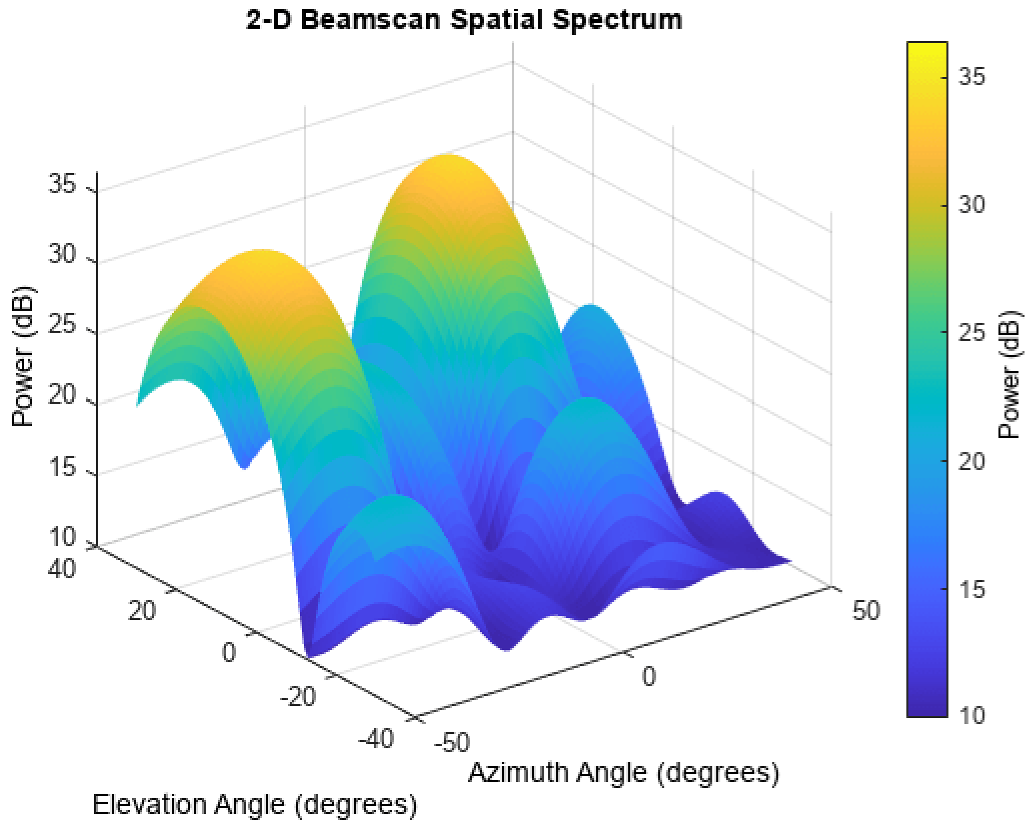
Because the values for the Azimuth`ScanAngles` and ElevationScanAngles properties have a granularity of 1°, the DOA estimates are not accurate. Improve the accuracy by choosing a finer grid

```
estimator2 = phased.BeamscanEstimator2D('SensorArray',array,'OperatingFrequency',fc, ...
    'DOAOutputPort',true,'NumSignals',2,'AzimuthScanAngles',-50:0.05:50,'ElevationScanAngles',-3
[~,doas] = estimator2(x);
disp(doas)
```

```
   17.3000  -37.4000
   20.5000   10.3000
```

Plot the beamscan spatial spectrum

```
plotSpectrum(estimator)
```



Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

phased.BeamscanEstimator | uv2azel | phitheta2azel

plotSpectrum

System object: `phased.BeamscanEstimator2D`
Package: `phased`

Plot spatial spectrum

Syntax

```
plotSpectrum(estimator)
plotSpectrum(estimator,Name,Value)
hl = plotSpectrum( ___ )
```

Description

`plotSpectrum(estimator)` plots the spatial spectrum resulting from the most recent execution of the object.

`plotSpectrum(estimator,Name,Value)` plots the spatial spectrum with additional options specified by one or more `Name,Value` pair arguments.

`hl = plotSpectrum( ___ )` returns the line handle in the figure.

Input Arguments

`H`

Spatial spectrum estimator object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`NormalizeResponse`

Set this value to `true` to plot the normalized spectrum. Setting this value to `false` plots the spectrum without normalization.

Default: `false`

`Title`

Character vector to use as figure title.

Default: `''`

Unit

Plot units, specified as `'db'`, `'mag'`, or `'pow'`.

Default: `'db'`

Examples

Estimate Directions of Arrival of Two Signals

Estimate the DOAs of two signals received by a 50-element URA with a rectangular lattice. The antenna operating frequency is 150 MHz. The actual direction of the first signal is -37° in azimuth and 0° in elevation. The direction of the second signal is 17° in azimuth and 20° in elevation.

```
antenna = phased.IsotropicAntennaElement('FrequencyRange',[100e6 300e6]);
array = phased.URA('Element',antenna,'Size',[5 10],'ElementSpacing',[1 0.6]);
fc = 150e6;
lambda = physconst('LightSpeed')/fc;
ang1 = [-37.5; 10.2];
ang2 = [17.4; 20.6];
x = sensorsig(getElementPosition(array)/lambda,8000,[ang1 ang2],0.2);
estimator = phased.BeamscanEstimator2D('SensorArray',array,'OperatingFrequency',fc, ...
    'DOAOutputPort',true,'NumSignals',2,'AzimuthScanAngles',-50:50,'ElevationScanAngles',-30:30)
[~,doas] = estimator(x);
disp(doas)
```

```
    17   -37
    20    10
```

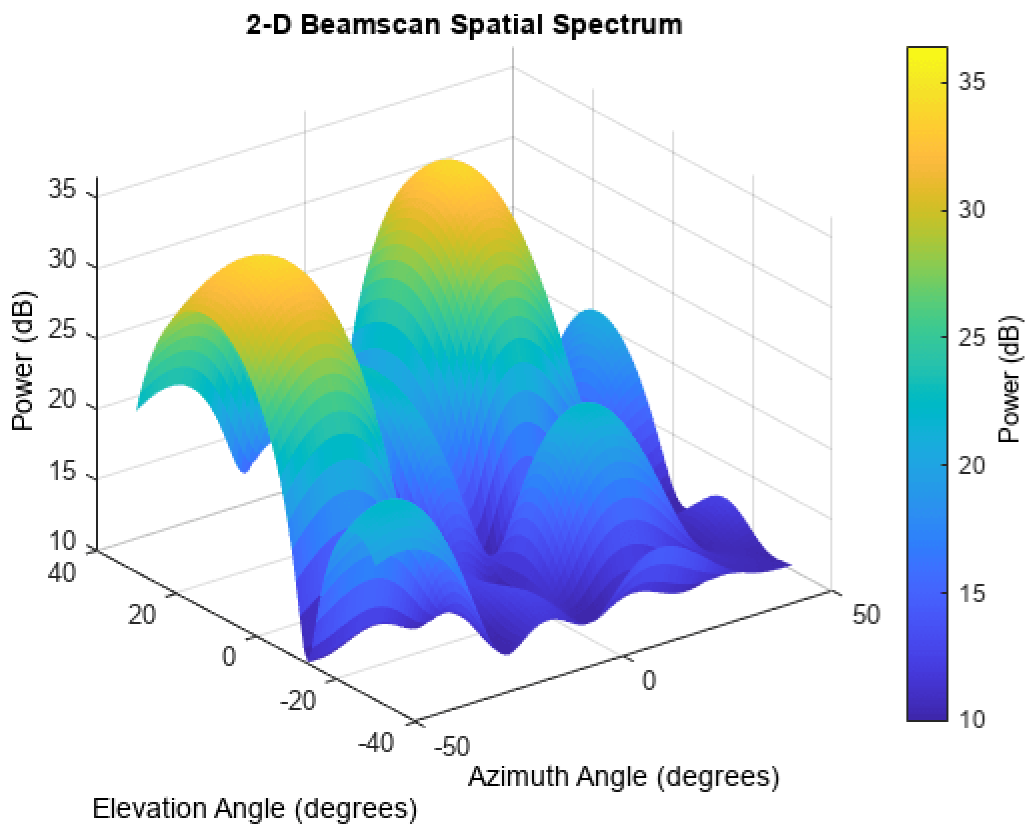
Because the values for the AzimuthScanAngles and ElevationScanAngles properties have a granularity of 1°, the DOA estimates are not accurate. Improve the accuracy by choosing a finer grid

```
estimator2 = phased.BeamscanEstimator2D('SensorArray',array,'OperatingFrequency',fc, ...
    'DOAOutputPort',true,'NumSignals',2,'AzimuthScanAngles',-50:0.05:50,'ElevationScanAngles',-3
[~,doas] = estimator2(x);
disp(doas)
```

```
   17.3000  -37.4000
   20.5000   10.3000
```

Plot the beamscan spatial spectrum

```
plotSpectrum(estimator)
```



2-D Beamscan Spatial Spectrum
Power (dB)
35
30
25
20
15
10
40
20
0
-20
-40
-50
0
50
Elevation Angle (degrees)
Azimuth Angle (degrees)
Power (dB)
35
30
25
20
15
10

reset

System object: `phased.BeamscanEstimator2D`
Package: `phased`

Reset states of 2-D beamscan spatial spectrum estimator object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `BeamscanEstimator2D` object, `H`.

step

System object: `phased.BeamscanEstimator2D`
Package: `phased`

Perform 2-D spatial spectrum estimation

Syntax

```
Y = step(H,X)
[Y,ANG] = step(H,X)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` estimates the spatial spectrum from `X` using the estimator `H`. `X` is a matrix whose columns correspond to channels. `Y` is a matrix representing the magnitude of the estimated 2-D spatial spectrum. `Y` has a row dimension equal to the number of elevation angles specified in `ElevationScanAngles` and a column dimension equal to the number of azimuth angles specified in `AzimuthScanAngles`. You can specify `X` as single or double precision.

`[Y,ANG] = step(H,X)` returns additional output `ANG` as the signal's direction of arrival (DOA) when the `DOAOutputPort` property is `true`. `ANG` is a two row matrix where the first row represents the estimated azimuth and the second row represents the estimated elevation (in degrees).

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Estimate DOAs of Two Sinusoidal Signals

Estimate the DOAs of two sinusoidal signals received by a 50-element URA with a rectangular lattice. The antenna operating frequency is 150 MHz. The actual direction of the first signal is -37° in azimuth and 0° in elevation. The direction of the second signal is 17° in azimuth and 20° in elevation.

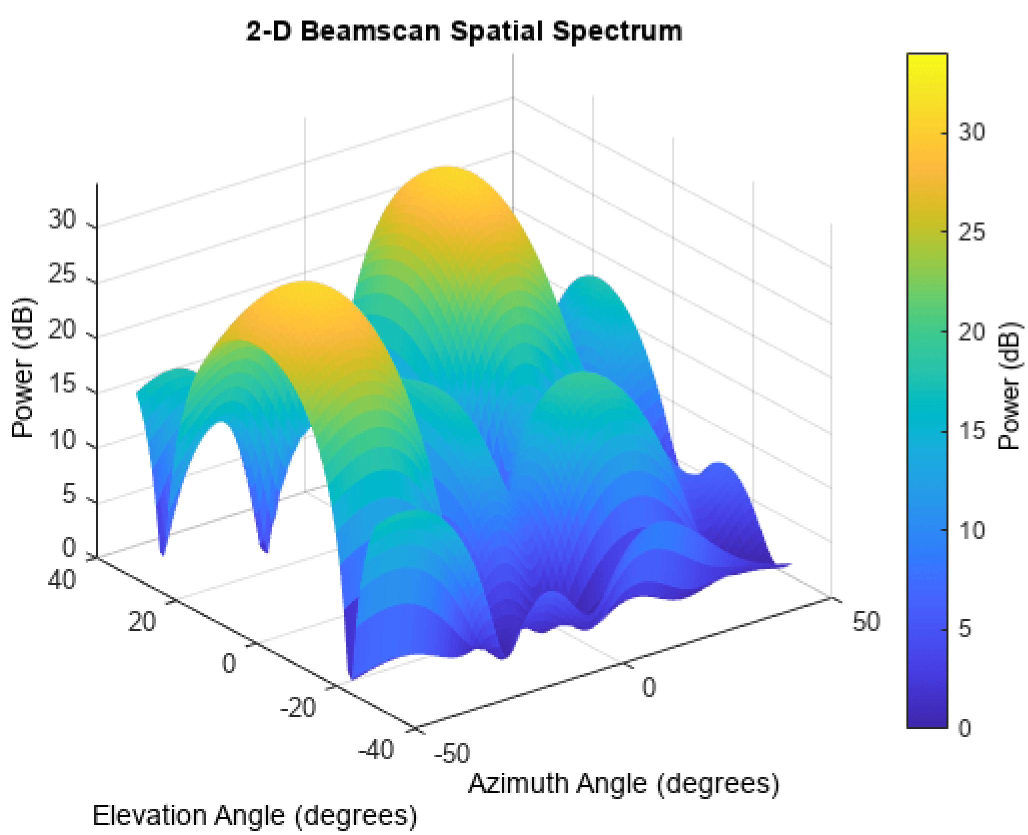
Create the signals and solve for the DOA's.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
array.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(array,[x1 x2],[-37 0; 17 20]',fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
estimator = phased.BeamscanEstimator2D('SensorArray',array, ...
    'OperatingFrequency',fc, ...
    'DOAOutputPort',true,'NumSignals',2, ...
    'AzimuthScanAngles',-50:50, ...
    'ElevationScanAngles',-30:30);
[~,doas] = estimator(x + noise)
```

```
doas = 2×2

   -37    17
     0    20
```

Plot the spatial spectrum.

```
plotSpectrum(estimator)
```



See Also

`azel2uv` | `azel2phitheta`

phased.BeamspaceESPRITEstimator

Package: `phased`

Beamspace ESPRIT direction of arrival (DOA) estimator for ULA

Description

The `BeamspaceESPRITEstimator` object computes a DOA estimate for a uniform linear array. The computation uses the estimation of signal parameters via rotational invariance techniques (ESPRIT) algorithm in beamspace.

To estimate the direction of arrival (DOA):

1. Define and set up your DOA estimator. See “Construction” on page 1-102.
2. Call `step` to estimate the DOA according to the properties of `phased.BeamspaceESPRITEstimator`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.BeamspaceESPRITEstimator` creates a beamspace ESPRIT DOA estimator System object, `H`. The object estimates the signal's direction of arrival using the beamspace ESPRIT algorithm with a uniform linear array (ULA).

`H = phased.BeamspaceESPRITEstimator(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be a `phased.ULA` object.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

`OperatingFrequency`

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

`SpatialSmoothing`

Spatial smoothing

Specify the number of averaging used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each additional smoothing handles one extra coherent source, but reduces the effective number of element by 1. The maximum value of this property is M–2, where M is the number of sensors. You can specify this property as single or double precision.

Default: `0`, indicating no spatial smoothing

`NumSignalsSource`

Source of number of signals

Specify the source of the number of signals as one of `'Auto'` or `'Property'`. If you set this property to `'Auto'`, the number of signals is estimated by the method specified by the `NumSignalsMethod` property. You can specify this property as single or double precision.

Default: `'Auto'`

`NumSignalsMethod`

Method to estimate number of signals

Specify the method to estimate the number of signals as one of `'AIC'` or `'MDL'`. `'AIC'` uses the Akaike Information Criterion and `'MDL'` uses Minimum Description Length Criterion. This property applies when you set the `NumSignalsSource` property to `'Auto'`.

Default: `'AIC'`

`NumSignals`

Number of signals

Specify the number of signals as a positive integer scalar. This property applies when you set the `NumSignalsSource` property to `'Property'`. You can specify this property as single or double precision.

Default: `1`

`Method`

Type of least square method

Specify the least squares method used for ESPRIT as one of `'TLS'` or `'LS'`. `'TLS'` refers to total least squares and `'LS'` refers to least squares.

Default: `'TLS'`

`BeamFanCenter`

Beam fan center direction (in degrees)

Specify the direction of the center of the beam fan (in degrees) as a real scalar value between –90 and 90. You can specify this property as single or double precision. This property is tunable.

Default: `0`

`NumBeamsSource`

Source of number of beams

Specify the source of the number of beams as one of `'Auto'` or `'Property'`. If you set this property to `'Auto'`, the number of beams equals N–L, where N is the number of array elements and L is the value of the `SpatialSmoothing` property.

Default: `'Auto'`

`NumBeams`

Number of beams

Specify the number of beams as a positive scalar integer. The lower the number of beams, the greater the reduction in computational cost. This property applies when you set the `NumBeamsSource` to `'Property'`. You can specify this property as single or double precision.

Default: `2`

Methods

step	Perform DOA estimation

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Estimate DOA of Two Signals Using Beamspace ESPRIT

Estimate the directions of arrival (DOA) of two signals received by a standard 10-element ULA with element spacing 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 45° in azimuth and 60° in elevation.

Create the two signals arriving at the array.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
array.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(array,[x1 x2],[10 20;45 60]',fc);
noise = 0.1/sqrt(2)*(randn(size(x)) + 1i*randn(size(x)));
```

Set up the beamspace ESPRIT estimator and solve for the DOAs.

```
estimator = phased.BeamspaceESPRITEstimator('SensorArray',array, ...
    'OperatingFrequency',fc,'NumSignalsSource','Property','NumSignals',2);
doas = estimator(x + noise);
az = broadside2az(sort(doas),[20 60])
```

```
az = 1×2

    9.9972   45.0061
```

Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`broadside2az` | `phased.ESPRITEstimator`

step

System object: `phased.BeamspaceESPRITEstimator`
Package: `phased`

Perform DOA estimation

Syntax

`ANG = step(H,X)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`ANG = step(H,X)` estimates the DOAs from `X` using the DOA estimator `H`. `X` is a matrix whose columns correspond to channels. `ANG` is a row vector of the estimated broadside angles (in degrees). You can specify the input data `X` as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Estimate DOA of Two Signals Using Beamspace ESPRIT

Estimate the directions of arrival (DOA) of two signals received by a standard 10-element ULA with element spacing 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 45° in azimuth and 60° in elevation.

Create the two signals arriving at the array.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
```

```
array.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(array,[x1 x2],[10 20;45 60]',fc);
noise = 0.1/sqrt(2)*(randn(size(x)) + 1i*randn(size(x)));
```

Set up the beamspace ESPRIT estimator and solve for the DOAs.

```
estimator = phased.BeamspaceESPRITEstimator('SensorArray',array, ...
    'OperatingFrequency',fc,'NumSignalsSource','Property','NumSignals',2);
doas = estimator(x + noise);
az = broadside2az(sort(doas),[20 60])
```

```
az = 1×2

    9.9972   45.0061
```

phased.CFARDetector

Package: `phased`

Constant false alarm rate (CFAR) detector

Description

The `CFARDetector` object implements a one-dimensional constant false-alarm rate (CFAR) detector. Detection processing is performed on selected elements (called cells) of the input data. A detection is declared when a cell value in the input data exceeds a threshold. To maintain a constant false alarm-rate, the threshold is set to a multiple of the local noise power of the input data. The detector estimates local noise power for a cell-under-test (CUT) from surrounding cells using one of three cell averaging methods, or an order statistics method. The cell-averaging methods are cell averaging (CA), greatest-of cell averaging (GOCA), or smallest-of cell averaging (SOCA).

For more information about CFAR detectors, see [1].

For each test cell, the detector:

1. estimates the noise statistic from the cell values in the training band surrounding the CUT cell.
2. computes the threshold by multiplying the noise estimate by the threshold factor.
3. compares the CUT cell value to the threshold to determine whether a target is present or absent. If the value is greater than the threshold, a target is present.

To run the detector

1. Define and set up your CFAR detector. See "Construction" on page 1-109.
2. Call `step` to perform CFAR detection according to the properties of `phased.CFARDetector`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.CFARDetector` creates a CFAR detector System object, `H`. The object performs CFAR detection on input data.

`H = phased.CFARDetector(Name,Value)` creates the object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`Method`

CFAR algorithm

Specify the CFAR detector algorithm as one of

`'CA'`	Cell-averaging CFAR
`'GOCA'`	Greatest-of cell-averaging CFAR
`'OS'`	Order statistic CFAR
`'SOCA'`	Smallest-of cell-averaging CFAR

Default: `'CA'`

`Rank`

Rank of order statistic

Specify the rank of the order statistic as a positive integer scalar. The value must be less than or equal to the value of the `NumTrainingCells` property. This property applies only when you set the `Method` property to `'OS'`. This property supports single and double precision,

Default: `1`

`NumGuardCells`

Number of guard cells

Specify the number of guard cells used in training as an even integer. This property specifies the total number of cells on both sides of the cell under test. This property supports single and double precision,

Default: `2`, indicating that there is one guard cell at both the front and back of the cell under test

`NumTrainingCells`

Number of training cells

Specify the number of training cells used in training as an even integer. Whenever possible, the training cells are equally divided before and after the cell under test. This property supports single and double precision.

Default: `2`, indicating that there is one training cell at both the front and back of the cell under test

`ThresholdFactor`

Methods of obtaining threshold factor

Specify whether the threshold factor comes from an automatic calculation, the `CustomThresholdFactor` property of this object, or an input argument in `step`. Values of this property are:

`'Auto'`	The application calculates the threshold factor automatically based on the desired probability of false alarm specified in the `ProbabilityFalseAlarm` property. The calculation assumes each independent signal in the input is a single pulse coming out of a square law detector with no pulse integration. The calculation also assumes the noise is white Gaussian.

'Custom'	The CustomThresholdFactor property of this object specifies the threshold factor.
'Input port'	An input argument in each invocation of step specifies the threshold factor.

Default: 'Auto'

ProbabilityFalseAlarm

Desired probability of false alarm

Specify the desired probability of false alarm as a scalar between 0 and 1 (not inclusive). This property applies only when you set the ThresholdFactor property to 'Auto'.

Default: 0.1

CustomThresholdFactor

Custom threshold factor

Specify the custom threshold factor as a positive scalar. This property applies only when you set the ThresholdFactor property to 'Custom'. This property is tunable. This property supports single and double precision,

Default: 1

OutputFormat

Format of detection results

Format of detection results returned by the step method, specified as 'CUT result' or 'Detection index'.

- When set to 'CUT result', the results are logical detection values (1 or 0) for each tested cell. 1 indicates that the value of the tested cell exceeds a detection threshold.
- When set to 'Detection index', the results form a vector or matrix containing the indices of tested cells which exceed a detection threshold. You can use this format as input to the phased.RangeEstimator and phased.DopplerEstimator System objects.

Default: 'CUT result'

ThresholdOutputPort

Output detection threshold

To obtain the detection threshold, set this property to true and use the corresponding output argument when invoking step. If you do not want to obtain the detection threshold, set this property to false.

Default: false

NoisePowerOutputPort

Output estimated noise

To obtain the estimated noise, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the estimated noise, set this property to `false`.

Default: `false`

`NumDetectionsSource`

Source of the number of detections

Source of the number of detections, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the number of detection indices reported is the total number of cells under test that have detections. If you set this property to `'Property'`, the number of reported detections is determined by the value of the `NumDetections` property.

Dependencies

To enable this property, set the `OutputFormat` property to `'Detection index'`.

Default: `'Auto'`

`NumDetections`

Maximum number of detections to report

Maximum number of detection indices to report, specified as a positive integer.

Dependencies

To enable this property, set the `OutputFormat` property to `'Detection index'` and the `NumDetectionsSource` property to `'Property'`.

Default: 1

Methods

step	Perform CFAR detection

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Compute PFA Using CFAR Detector on Noise

Perform cell-averaging CFAR detection on a given Gaussian noise vector with a desired probability of false alarm (pfa) of 0.1. Assume that the data comes from a square law detector and no pulse integration is performed. Use 50 cells to estimate the noise level and 1 cell to separate the test cell and training cells. Perform the detection on all cells of the input.

```
detector = phased.CFARDetector('NumTrainingCells',50,...
    'NumGuardCells',2,'ProbabilityFalseAlarm',0.1);
N = 1000;
x = 1/sqrt(2)*(randn(N,1) + 1i*randn(N,1));
```

```
dets = detector(abs(x).^2,1:N);
pfa = sum(dets)/N

pfa = 0.1140
```

Compute CFAR Detection Indices

Perform cell-averaging CFAR detection on a given Gaussian noise vector with a desired probability of false alarm (pfa) of 0.005. Assume that the data comes from a square law detector and no pulse integration is performed. Perform the detection on all cells of the input. Use 50 cells to estimate the noise level and 1 cell to separate the test cell and training cells. Display the detection indices.

```
rng default;
detector = phased.CFARDetector('NumTrainingCells',50,'NumGuardCells',2, ...
    'ProbabilityFalseAlarm',0.005,'OutputFormat','Detection index');
N = 1000;
x1 = 1/sqrt(2)*(randn(N,1) + 1i*randn(N,1));
x2 = 1/sqrt(2)*(randn(N,1) + 1i*randn(N,1));
x = [x1,x2];
cutidx = 1:N;
dets = detector(abs(x).^2,cutidx)

dets = 2×11

   339   537   538   734   786   827   979   136   418   539   874
     1     1     1     1     1     1     1     2     2     2     2
```

Algorithms

CFAR Detection

`phased.CFARDetector` uses cell averaging in three steps:

1. Identify the training cells from the input, and form the noise estimate. The next table indicates how the detector forms the noise estimate, depending on the `Method` property value.

Method	Noise Estimate
'CA'	Use the average of the values in all the training cells.
'GOCA'	Select the greater of the averages in the front training cells and rear training cells.
'OS'	Sort the values in the training cells in ascending order. Select the Nth item, where N is the value of the `Rank` property.
'SOCA'	Select the smaller of the averages in the front training cells and rear training cells.

2. Multiply the noise estimate by the threshold factor to form the threshold.
3. Compare the value in the test cell against the threshold to determine whether the target is present or absent. If the value is greater than the threshold, the target is present.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`npwgnthresh` | `phased.MatchedFilter` | `phased.TimeVaryingGain`

step

System object: `phased.CFARDetector`
Package: `phased`

Perform CFAR detection

Syntax

```
Y = step(H,X,cutidx)
[Y,th] = step( ___ )
[Y,noise] = step( ___ )
Y = step(H,X,cutidx,thfac)
[Y,TH,N] = step(H,X,cutidx,thfac)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X,cutidx)` performs CFAR detection on specified elements of the input data, `X`. `X` can either be a real-valued *M*-by-1 column vector or a real-valued *M*-by-*N* matrix. `cutidx` is a length-*D* vector of indices specifying the input elements or cells under test (CUT) on which to perform detection processing. When `X` is a vector, `cutidx` specifies the element. When `X` is a matrix, `cutidx` specifies the row of the element. The same index applies to all columns of the matrix. Detection is performed independently along each column of `X` for the indices specified in `cutidx`. You can specify the input arguments as single or double precision.

The output argument `Y` contains detection results. The format of `Y` depends on the `OutputFormat` property.

- When `OutputFormat` is `'Cut result'`, `Y` is a *D*-by-1 vector or a *D*-by-*N* matrix containing logical detection results. *D* is the length of `cutidx` and *N* is the number of columns of `X`. The rows of `Y` correspond to the rows in `cutidx`. For each row, `Y` contains `1` in a column if there is a detection in the corresponding column of `X`. Otherwise, `Y` contains a `0`.
- When `OutputFormat` is `'Detection index'`, `Y` is a *1*-by-*L* vector or a 2-by-*L* matrix containing detections indices. *L* is the number of detections found in the input data. When `X` is a column vector, `Y` contains the index for each detection in `X`. When `X` is a matrix, `Y` contains the row and column indices of each detection in `X`. Each column of `Y` has the form `[detrow;detcol]`. When the `NumDetectionsSource` property is set to `'Property'`, *L* equals the value of the `NumDetections` property. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`[Y,th] = step( ___ )` also returns the detection threshold, `th`, applied to detected cells under test.

- When `OutputFormat` is `'CUT result'`, `th` also returns the detection threshold `th`.
- When `OutputFormat` is `'Detection index'`, `th` returns a detection threshold for each corresponding detection in `Y`. When the `NumDetectionsSource` property is set to `'Property'`, *L* equals the value of the `NumDetections` property. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

To enable this syntax, set the `ThresholdOutputPort` property to `true`.

`[Y,noise] = step( ___ )` also returns the estimated noise power, `noise`, for each detected cell under test in `X`.

- When `OutputFormat` is `'CUT result'`, `noise` returns a noise power estimate.
- When `OutputFormat` is `'Detection index'`, `noise` returns a noise power estimate for each corresponding detection in `Y`. When the `NumDetectionsSource` property is set to `'Property'`, *L* equals the value of the `NumDetections` property. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

To enable this syntax, set the `NoisePowerOutputPort` property to `true`.

`Y = step(H,X,cutidx,thfac)`, in addition, specifies `thfac` as the threshold factor used to calculate the detection threshold. `thfac` must be a positive scalar. To enable this syntax, set the `ThresholdFactor` property to `'Input port'`.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example, `[Y,TH,N] = step(H,X,cutidx,thfac)`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Compute PFA Using CFAR Detector on Noise

Perform cell-averaging CFAR detection on a given Gaussian noise vector with a desired probability of false alarm (pfa) of 0.1. Assume that the data comes from a square law detector and no pulse integration is performed. Use 50 cells to estimate the noise level and 1 cell to separate the test cell and training cells. Perform the detection on all cells of the input.

```
detector = phased.CFARDetector('NumTrainingCells',50,...
    'NumGuardCells',2,'ProbabilityFalseAlarm',0.1);
N = 1000;
x = 1/sqrt(2)*(randn(N,1) + 1i*randn(N,1));
dets = detector(abs(x).^2,1:N);
pfa = sum(dets)/N
```

```
pfa = 0.1140
```

Compute CFAR Detection Indices

Perform cell-averaging CFAR detection on a given Gaussian noise vector with a desired probability of false alarm (pfa) of 0.005. Assume that the data comes from a square law detector and no pulse integration is performed. Perform the detection on all cells of the input. Use 50 cells to estimate the noise level and 1 cell to separate the test cell and training cells. Display the detection indices.

```
rng default;
detector = phased.CFARDetector('NumTrainingCells',50,'NumGuardCells',2, ...
    'ProbabilityFalseAlarm',0.005,'OutputFormat','Detection index');
N = 1000;
x1 = 1/sqrt(2)*(randn(N,1) + 1i*randn(N,1));
x2 = 1/sqrt(2)*(randn(N,1) + 1i*randn(N,1));
x = [x1,x2];
cutidx = 1:N;
dets = detector(abs(x).^2,cutidx)
```

```
dets = 2×11

   339   537   538   734   786   827   979   136   418   539   874
     1     1     1     1     1     1     1     2     2     2     2
```

Algorithms

`phased.CFARDetector` uses cell averaging in three steps:

1. Identify the training cells from the input, and form the noise estimate. The next table indicates how the detector forms the noise estimate, depending on the `Method` property value.

Method	Noise Estimate
`'CA'`	Use the average of the values in all the training cells.
`'GOCA'`	Select the greater of the averages in the front training cells and rear training cells.
`'OS'`	Sort the values in the training cells in ascending order. Select the *N*th item, where *N* is the value of the `Rank` property.
`'SOCA'`	Select the smaller of the averages in the front training cells and rear training cells.

2. Multiply the noise estimate by the threshold factor to form the threshold.
3. Compare the value in the test cell against the threshold to determine whether the target is present or absent. If the value is greater than the threshold, the target is present.

For details, see [1].

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

phased.CFARDetector2D

Package: `phased`

Two-dimensional CFAR detector

Description

`phased.CFARDetector2D` System object implements a constant false-alarm rate detector (CFAR) for selected elements (called cells) of two-dimensional image data. A detection is declared when an image cell value exceeds a threshold. To maintain a constant false alarm-rate, the threshold is set to a multiple of the image noise power. The detector estimates noise power for a cell-under-test (CUT) from surrounding cells using one of three cell averaging methods, or an order statistics method. The cell-averaging methods are cell averaging (CA), greatest-of cell averaging (GOCA), or smallest-of cell averaging (SOCA).

For each test cell, the detector:

1. estimates the noise statistic from the cell values in the training band surrounding the CUT cell.
2. computes the threshold by multiplying the noise estimate by the threshold factor.
3. compares the CUT cell value to the threshold to determine whether a target is present or absent. If the value is greater than the threshold, a target is present.

To run the detector

1. Define and set up your 2-D CFAR detector. You can set the `phased.CFARDetector2D` System object properties when you create the object, or leave them set to their default values. See “Construction” on page 1-118. Some properties that you set at construction time can be changed later. These properties are *tunable*.
2. Find the detections by calling the `step` method. The output of this method depends on the properties of the `phased.CFARDetector2D` System object.

Note Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`detector = phased.CFARDetector2D` creates a 2-D CFAR detector System object, `detector`.

`detector = phased.CFARDetector2D(Name,Value)` creates a 2-D CFAR System object, `detector`, with each specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`Method` — Two-dimensional CFAR averaging method

`'CA'` (default) | `'GOCA'` | `'SOCA'` | `'OS'`

Two-dimensional CFAR averaging method, specified as `'CA'`, `'GOCA'`, `'SOCA'`, or `'OS'`. For `'CA'`, `'GOCA'`, `'SOCA'`, the noise power is the sample mean derived from the training band. For `'OS'`, the noise power is the *k*th cell value obtained from numerically ordering all training cell values. Set *k* using the `Rank` property.

Averaging Method	Description
`CA` — Cell-averaging algorithm	Computes the sample mean of all training cells surrounding the CUT cell.
`GOCA` — Greatest-of cell-averaging algorithm	Splits the 2-D training window surrounding the CUT cell into left and right halves. Then, the algorithm computes the sample mean for each half and selects the largest mean.
`SOCA` — Smallest-of cell-averaging algorithm	Splits the 2-D training window surrounding the CUT cell into left and right halves. Then, the algorithm computes the sample mean for each half and selects the smallest mean.
`OS` — Order statistic algorithm	Sorts training cells in ascending order of numeric values. Then the algorithm selects the *k*th value from the list. *k* is the rank specified by the `Rank` parameter.

Example: `'OS'`

Data Types: `char`

`GuardBandSize` — Widths of guard band
`[1 1]` (default) | nonnegative integer | 2-element vector of nonnegative integers

The number of rows and columns of the guard band cells on each side of the CUT cell, specified as nonnegative integers. The first element specifies the guard band size along the row dimension. The second element specifies the guard band size along the column dimension. Specifying this property as a single integer is equivalent to specifying a guard band with the same value for both dimensions. For example, a value of `[1 1]`, indicates that there is a one guard-cell-wide region surrounding each CUT cell. A value of zero indicates there are no guard cells.

Example: `[2 3]`

Data Types: `single` | `double`

`TrainingBandSize` — Widths of training band
`[1 1]` (default) | positive integer | 2-element vector of positive integers

The number of rows and columns of the training band cells on each side of the CUT cell, specified as a positive integer or a 1-by-2 matrix of positive integers. The first element specifies the training band size along the row dimension. The second element specifies the training band size along the column dimension. Specifying this property as a scalar is equivalent to specifying a training band with the same value for both dimensions. For example, a value of `[1 1]` indicates a 1 training-cell-wide region surrounding the CUT cell.

Example: `[-30:0.1:30]`

Data Types: `single` | `double`

`Rank` — Rank of order statistic
1 (default) | positive integer

Rank of the order statistic used in the 2-D CFAR algorithm, specified as a positive integer. The value of this property must lie between 1 and N_{train}, where N_{train} is the number of training cells. A value of 1 selects the smallest value in the training region.

Example: `5`

Dependencies

To enable this property, set the `Method` property to `'OS'`.

Data Types: `single` | `double`

`ThresholdFactor` — Threshold factor method
`'Auto'` (default) | `'Input port'` | `'Custom'`

Threshold factor method, specified as `'Auto'`, `'Input port'`, or `'Custom'`.

When you set the `ThresholdFactor` property to `'Auto'`, the threshold factor is calculated from the desired probability of false alarm set in the `ProbabilityFalseAlarm` property. The calculation assumes that each independent signal in the input is a single pulse coming out of a square law detector with no pulse integration. In addition, the noise is assumed to be white Gaussian.

When you set the `ThresholdFactor` property to `'Input port'`, the threshold factor is obtained from an input argument of the `step` method.

When you set the `ThresholdFactor` property to `'Custom'`, the threshold factor is obtained from the value of the `CustomThresholdFactor` property.

Example: `'Custom'`

Data Types: `char`

`ProbabilityFalseAlarm` — Required probability of false alarm
`0.1` (default) | positive scalar between 0 and 1

Required probability of false alarm, specified as a real positive scalar between 0 and 1. The algorithm calculates the threshold factor from the required probability of false alarm.

Example: `0.001`

Dependencies

To enable this property, set the `ThresholdFactor` property to `'Auto'`.

Data Types: `single` | `double`

`CustomThresholdFactor` — Custom threshold factor
`1` (default) | positive scalar

Custom threshold factor, specified as a real positive scalar. This property is tunable.

Dependencies

To enable this property, set the `ThresholdFactor` property to `'Custom'`.

Data Types: `single` | `double`

`OutputFormat` — Format of detection results
`'CUT result'` (default) | `'Detection index'`

Format of detection results returned by the `step` method, specified as `'CUT result'` or `'Detection index'`.

- When set to `'CUT result'`, the results are logical detection values (`1` or `0`) for each tested cell.
- When set to `'Detection index'`, the results form a vector or matrix containing the indices of tested cells that exceed a detection threshold. You can use this format as input to the `phased.RangeEstimator` and `phased.DopplerEstimator` System objects.

Data Types: `char`

`ThresholdOutputPort` — Enable detection threshold output
`false` (default) | `true`

Option to enable detection threshold output, specified as `false` or `true`. Setting this property to `true` returns the detection threshold via an output argument, `th`, of the `step` method.

Data Types: `logical`

`NoisePowerOutputPort` — Enable noise power output
`false` (default) | `true`

Option to enable output of noise power, specified as `false` or `true`. Setting this property to `true` returns the noise power via the output argument, `noise`, of the `step` method.

Data Types: `logical`

`NumDetectionsSource` — Source of the number of detections
`'Auto'` (default) | `'Property'`

Source of the number of detections, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the number of detection indices reported is the total number of cells under test that have detections. If you set this property to `'Property'`, the number of reported detections is determined by the value of the `NumDetections` property.

Dependencies

To enable this property, set the `OutputFormat` property to `'Detection index'`.

Data Types: `char`

`NumDetections` — Maximum number of detection indices to report
`1` (default) | positive integer

Maximum number of detection indices to report, specified as a positive integer.

Dependencies

To enable this property, set the `OutputFormat` property to `'Detection index'` and the `NumDetectionsSource` property to `'Property'`.

Data Types: `double`

Methods

reset	Reset states of System object
step	Two-dimensional CFAR detection

Common to All System Objects	
release	Allow System object property value changes

Examples

Set 2-D CFAR Threshold for Noise-Only Data

This example shows how to set a 2-D CFAR threshold based upon a required probability of false alarm (pfa).

Perform cell-averaging CFAR detection on a 41-by-41 matrix of cells containing Gaussian noise. Estimate the empirical pfa and compare it to the required pfa. To get a good estimate, perform this simulation on 1000 similar matrices. First, set a threshold using the required pfa. In this case, there are no targets and the pfa can be estimated from the number of cells that exceed the threshold. Assume that the data is processed through a square-law detector and that no pulse integration is performed. Use a training-cell band of 3 cells in width and 4 cells in height. Use a guard band of 3 cells in width and 2 cells in height to separate the cells under test (CUT) from the training cells. Specify a required pfa of 5.0e-4.

```
p = 5e-4;
rs = RandStream.create('mt19937ar','Seed',5);
N = 41;
ntrials = 1000;
detector = phased.CFARDetector2D('TrainingBandSize',[4,3], ...
    'ThresholdFactor','Auto','GuardBandSize',[2,3], ...
    'ProbabilityFalseAlarm',p,'Method','SOCA','ThresholdOutputPort',true);
```

Create a 41-by-41 image containing random complex data. Then, square the data to simulate a square-law detector.

```
x = 2/sqrt(2)*(randn(rs,N,N,ntrials) + 1i*randn(rs,N,N,ntrials));
x2 = abs(x).^2;
```

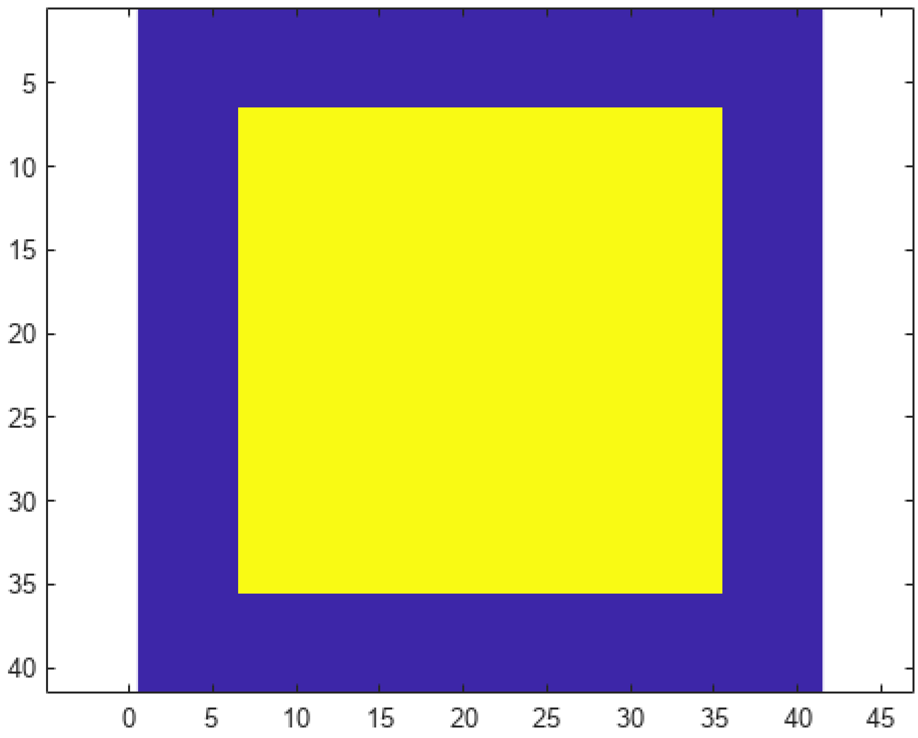
Process all the cells in each image. To do this, find the row and column of each CUT cell whose training region falls entirely within each image.

```
Ngc = detector.GuardBandSize(2);
Ngr = detector.GuardBandSize(1);
Ntc = detector.TrainingBandSize(2);
Ntr = detector.TrainingBandSize(1);
cutidx = [];
colstart = Ntc + Ngc + 1;
colend = N - ( Ntc + Ngc);
rowstart = Ntr + Ngr + 1;
rowend = N - ( Ntr + Ngr);
for m = colstart:colend
    for n = rowstart:rowend
        cutidx = [cutidx,[n;m]];
```

```
    end
end
ncutcells = size(cutidx,2);
```

Display the CUT cells.

```
cutimage = zeros(N,N);
for k = 1:ncutcells
    cutimage(cutidx(1,k),cutidx(2,k)) = 1;
end
imagesc(cutimage)
axis equal
```



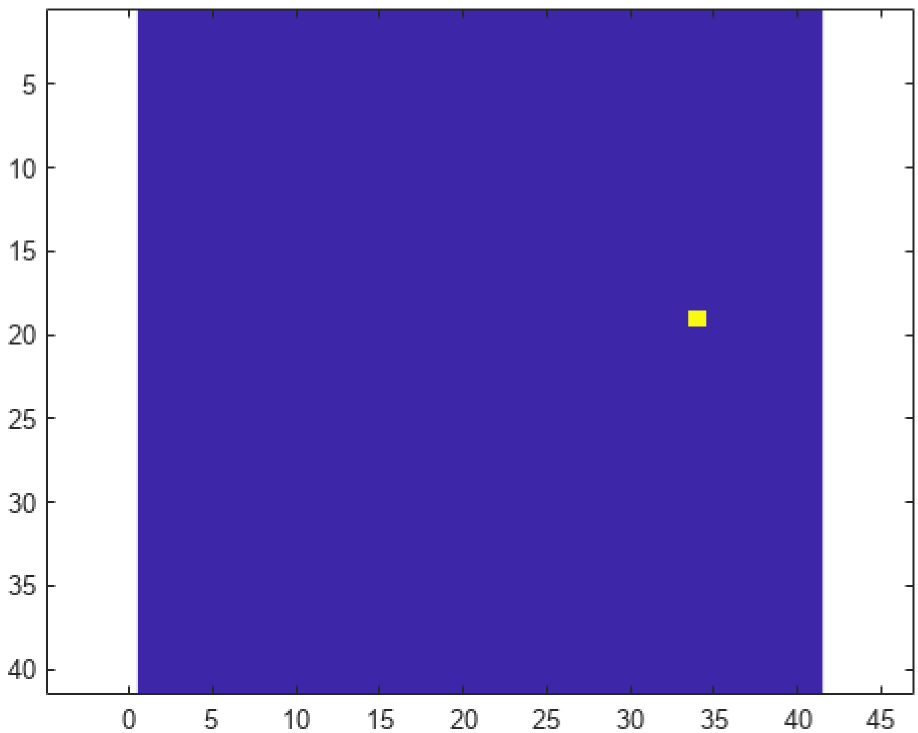
Perform the detection on all CUT cells. Return the detection classification and the threshold used to classify the cell.

```
[dets,th] = detector(x2,cutidx);
```

Find and display an image with a false alarm for illustration.

```
di = [];
for k = 1:ntrials
    d = dets(:,k);
    if (any(d) > 0)
        di = [di,k];
    end
end
idx = di(1);
```

```
detimg = zeros(N,N);
for k = 1:ncutcells
    detimg(cutidx(1,k),cutidx(2,k)) = dets(k,idx);
end
imagesc(detimg)
axis equal
```



Compute the empirical pfa.

```
pfa = sum(dets(:))/ntrials/ncutcells
```

```
pfa = 4.5898e-04
```

The empirical and specified pfa agree.

Display the average empirical threshold value over all images.

```
mean(th(:))
```

```
ans = 31.7139
```

Compute the theoretical threshold factor for the required pfa.

```
threshfactor = npwgnthresh(p,1,'noncoherent');
threshfactor = 10^(threshfactor/10);
disp(threshfactor)
```

```
    7.6009
```

The theoretical threshold factor multiplied by the noise variance should agree with the measured threshold.

```
noisevar = mean(x2(:));
disp(threshfactor*noisevar);
```

```
   30.4118
```

The theoretical threshold and empirical threshold agree to within an acceptable difference.

Detect Targets in Background Noise

Perform cell-averaging CFAR detection on a 41-by-41 matrix of cells containing five closely-spaced targets in Gaussian noise. Perform this detection on a simulation of 1000 images. Use two detectors with different guard band regions. Set the thresholds manually using the `Custom` threshold factor. Assume that the data is processed through a square law-detector and that no pulse integration is performed. Use a training cell band of 2 cells in width and 2 cells in height. For the first detector, use a guard band of 1 cell all around to separate the CUT cells from the training cells. For the second detector, use a guard band of 8 cells all around.

```
p = 5e-4;
rs = RandStream.create('mt19937ar','Seed',5);
N = 41;
ntrials = 1000;
```

Create 1000 41-by-41 images of complex random noise with standard deviation of 1.

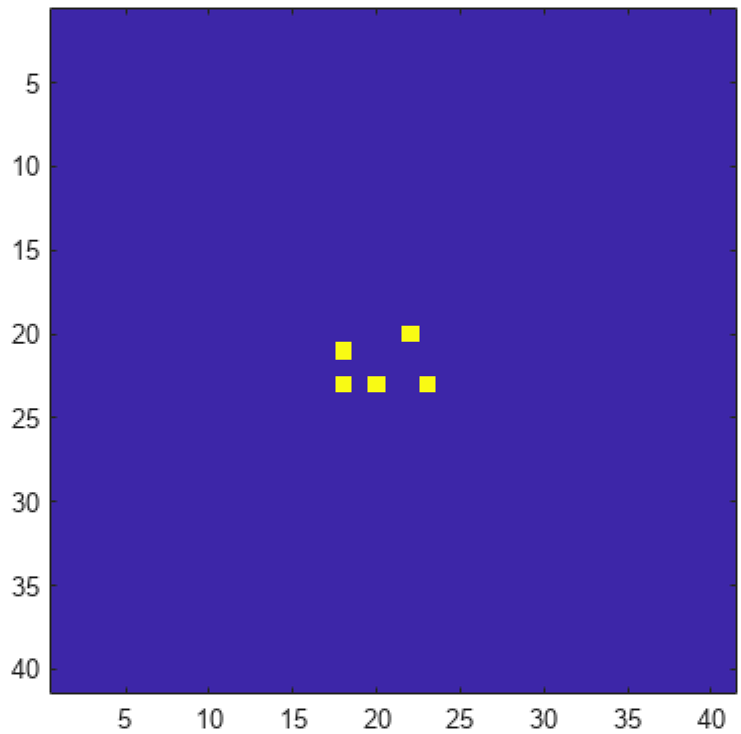
```
s = 1;
x = s/sqrt(2)*(randn(rs,N,N,ntrials) + 1i*randn(rs,N,N,ntrials));
```

Set the target cells values to 1.5. Then, square the cell values.

```
A = 1.5;
x(23,20,:) = A;
x(23,18,:) = A;
x(23,23,:) = A;
x(20,22,:) = A;
x(21,18,:) = A;
x2 = abs(x).^2;
```

Display the target cells.

```
xtgt = zeros(N,N);
xtgt(23,20,:) = A;
xtgt(23,18,:) = A;
xtgt(23,23,:) = A;
xtgt(20,22,:) = A;
xtgt(21,18,:) = A;
imagesc(xtgt)
axis equal
axis tight
```



Set the CUT cells to be the target cells.

```
cutidx(1,1) = 23;
cutidx(2,1) = 20;
cutidx(1,2) = 23;
cutidx(2,2) = 18;
cutidx(1,3) = 23;
cutidx(2,3) = 23;
cutidx(1,4) = 20;
cutidx(2,4) = 22;
cutidx(1,5) = 21;
cutidx(2,5) = 18;
```

Perform the detection on all CUT cells using two CFAR 2-D detectors. The first detector has a small guard band region. The training region can include neighboring targets which can affect the computation of the noise power. The second detector has a larger guard band region, which precludes target cells from being used in the noise computation.

Create the two CFAR detectors.

```
detector1 = phased.CFARDetector2D('TrainingBandSize',[2,2], ...
    'GuardBandSize',[1,1],'ThresholdFactor','Custom','Method','CA', ...
    'CustomThresholdFactor',2,'ThresholdOutputPort',true);
detector2 = phased.CFARDetector2D('TrainingBandSize',[2,2], ...
    'GuardBandSize',[8,8],'ThresholdFactor','Custom','Method','CA', ...
    'CustomThresholdFactor',2,'ThresholdOutputPort',true);
```

Return the detection classifications and the thresholds used to classify the cells. Then, compute the probabilities of detection.

```
[dets1,th1] = detector1(x2,cutidx);
ndets = numel(dets1(:));
pd1 = sum(dets1(:))/ndets
```

```
pd1 = 0.6416
```

```
[dets2,th2] = detector2(x2,cutidx);
pd2 = sum(dets2(:))/ndets
```

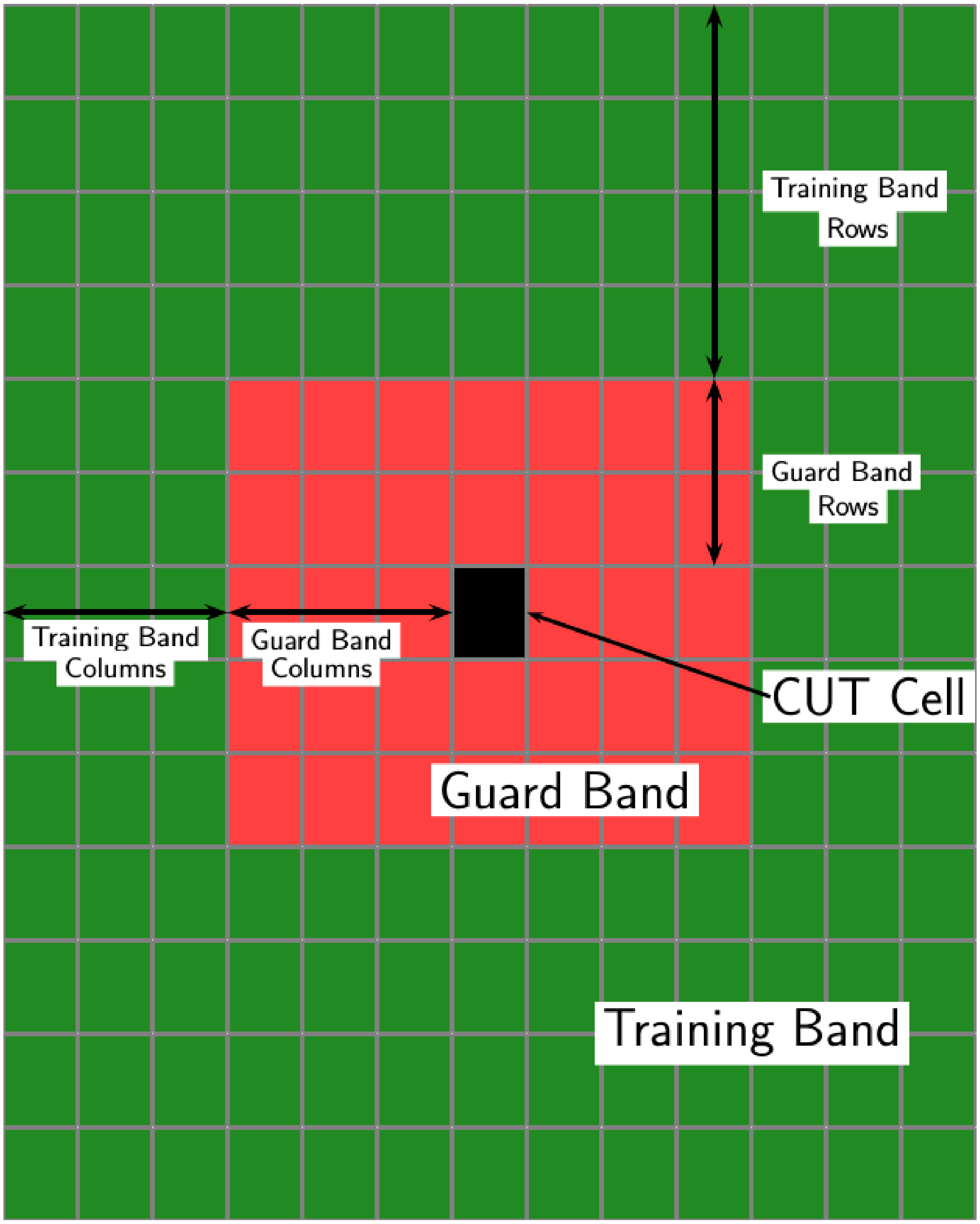
```
pd2 = 0.9396
```

The detector with the larger guard-band region has a higher pfa because the noise is more accurately estimated.

More About

Training Cells

CFAR 2-D requires an estimate of the noise power. Noise power is computed from cells that are assumed not to contain any target signal. These cells are the training cells. Training cells form a band around the cell-under-test (CUT) cell but may be separated from the CUT cell by a guard band. The detection threshold is computed by multiplying the noise power by the threshold factor.



For GOCA and SOCA averaging, the noise power is derived from the mean value of one of the left or right halves of the training cell region.

Because the number of columns in the training region is odd, the cells in the middle column are assigned equally to either the left or right half.

When using the order-statistic method, the rank cannot be larger than the number of cells in the training cell region, N_{train}. You can compute N_{train}.

- N_{TC} is the number of training band columns.

- N_{TR} is the number of training band rows.
- N_{GC} is the number of guard band columns.
- N_{GR} is the number of guard band rows.

The total number of cells in the combined training region, guard region, and CUT cell is $N_{total} = (2N_{TC} + 2N_{GC} + 1)(2N_{TR} + 2N_{GR} + 1)$.

The total number of cells in the combined guard region and CUT cell is $N_{guard} = (2N_{GC} + 1)(2N_{GR} + 1)$.

The number of training cells is $N_{train} = N_{total} - N_{guard}$.

By construction, the number of training cells is always even. Therefore, to implement a median filter, you can choose a rank of $N_{train}/2$ or $N_{train}/2 + 1$.

Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2016b

References

[1] Mott, H. *Antennas for Radar and Communications*. New York: John Wiley & Sons, 1992.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[3] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

Functions
`npwgnthresh` | `rocpfa`

Objects
`phased.CFARDetector`

Blocks
2-D CFAR Detector | CFAR Detector

Topics
"Modeling Target Radar Cross Section" (Radar Toolbox)
"Simulating Test Signals for a Radar Receiver"

reset

System object: phased.CFARDetector2D
Package: phased

Reset states of System object

Syntax

```
reset(detector)
```

Description

`reset(detector)` resets the internal state of the `phased.CFARDetector2D`object, `detector`.

Input Arguments

`detector` — Two-dimensional CFAR detector
`phased.CFARDetector2D` System object

Two-dimensional CFAR detector, specified as a `phased.CFARDetector2D` System object.

Version History

Introduced in R2016b

step

System object: `phased.CFARDetector2D`
Package: `phased`

Two-dimensional CFAR detection

Syntax

```
Y = step(detector,X,cutidx)
Y = step(detector,X,cutidx,K)
[Y,th] = step( ___ )
[Y,noise] = step( ___ )
```

Description

Note Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(detector,X,cutidx)` performs 2-D CFAR detection on input image data, `X`, for the image cells under test (CUT) specified by `cutidx`. `Y` contains the detection results for the CUT cells.

`Y = step(detector,X,cutidx,K)` also specifies a threshold factor, `K`, for setting the detection threshold. This syntax applies when the `ThresholdFactor` property of the detector is set to `'Input port'`.

`[Y,th] = step( ___ )` also returns the detection threshold, `th`, applied to detected cells under test. To enable this syntax, set the `ThresholdOutputPort` property to `true`.

`[Y,noise] = step( ___ )` also returns the estimated noise power, `noise`, applied to detected cells under test. To enable this syntax, set the `NoisePowerOutputPort` property to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`detector` — Two-dimensional CFAR detector

`phased.CFARDetector2D` System object

Two-dimensional CFAR detector, specified as a `phased.CFARDetector2D` System object.

`X` — Input image

real *M*-by-*N* matrix | real *M*-by-*N*-by-*P* array

Input image, specified as a real *M*-by-*N* matrix or a real *M*-by-*N*-by-*P* array. *M* and *N* represent the rows and columns of a matrix. Each page is an independent 2-D signal.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;2.5,1;0.5,0.1]`

Data Types: `single` | `double`

`cutidx` — Test cells
2-by-*D* matrix of positive integers

Test cells, specified as a *2*-by-*D* matrix of positive integers, where *D* is the number of test cells. Each column of `cutidx` specifies the row and column indices of a CUT cell. The same indices apply to all pages in the input array. You must restrict the locations of CUT cells so that their training regions lie completely within the input images.

Example: `[10,15;11,15;12,15]`

Data Types: `single` | `double`

`K` — Detection threshold factor
positive scalar

Threshold factor used to calculate the detection threshold, specified as a positive scalar.

Dependencies

To enable this input argument, set the `ThresholdFactor` property of the detector object to `'Input port'`

Data Types: `single` | `double`

Output Arguments

`Y` — Detection results
L-by-*P* logical matrix

Detection results, whose format depends on the `OutputFormat` property

- When `OutputFormat` is `'Cut result'`, `Y` is a *D*-by-*P* matrix containing logical detection results for cells under test. *D* is the length of `cutidx` and *P* is the number of pages of `X`. The rows of `Y` correspond to the rows of `cutidx`. For each row, `Y` contains `1` in a column if there is a detection in the corresponding cell in `X`. Otherwise, `Y` contains a `0`.
- When `OutputFormat` is `'Detection report'`, `Y` is a *K*-by-*L* matrix containing detections indices. *K* is the number of dimensions of `X`. *L* is the number of detections found in the input data. When `X` is a matrix, `Y` contains the row and column indices of each detection in `X` in the form `[detrow;detcol]`. When `X` is an array, `Y` contains the row, column, and page indices of each detection in `X` in the form `[detrow;detcol;detpage]`. When the `NumDetectionsSource` property is set to `'Property'`, *L* equals the value of the `NumDetections` property. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

Data Types: `single` | `double`

`th` — Computed detection threshold
real-valued matrix

Computed detection threshold for each detected cell, returned as a real-valued matrix. `Th` has the same dimensions as `Y`.

- When `OutputFormat` is `'CUT result'`, `Th` returns the detection threshold whenever an element of `Y` is `1` and `NaN` whenever an element of `Y` is `0`.
- When `OutputFormat` is `'Detection index'`, `th` returns a detection threshold for each corresponding detection in `Y`. When the `NumDetectionsSource` property is set to `'Property'`, *L* equals the value of the `NumDetections` property. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

Dependencies

To enable this output argument, set the `ThresholdOutputPort` to `true`.

Data Types: `single` | `double`

`noise` — Estimated noise power
real-valued matrix

Estimated noise power for each detected cell, returned as a real-valued matrix. `noise` has the same dimensions as `Y`.

- When `OutputFormat` is `'CUT result'`, `noise` returns the noise power whenever an element of `Y` is `1` and `NaN` whenever an element of `Y` is `0`.
- When `OutputFormat` is `'Detection index'`, `noise` returns a noise power for each corresponding detection in `Y`. When the `NumDetectionsSource` property is set to `'Property'`, *L* equals the value of the `NumDetections` property. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

Dependencies

To enable this output argument, set the `NoisePowerOutputPort` to `true`.

Data Types: `single` | `double`

Examples

Set 2-D CFAR Threshold for Noise-Only Data

This example shows how to set a 2-D CFAR threshold based upon a required probability of false alarm (pfa).

Perform cell-averaging CFAR detection on a 41-by-41 matrix of cells containing Gaussian noise. Estimate the empirical pfa and compare it to the required pfa. To get a good estimate, perform this simulation on 1000 similar matrices. First, set a threshold using the required pfa. In this case, there are no targets and the pfa can be estimated from the number of cells that exceed the threshold. Assume that the data is processed through a square-law detector and that no pulse integration is performed. Use a training-cell band of 3 cells in width and 4 cells in height. Use a guard band of 3 cells in width and 2 cells in height to separate the cells under test (CUT) from the training cells. Specify a required pfa of 5.0e-4.

```
p = 5e-4;
rs = RandStream.create('mt19937ar','Seed',5);
N = 41;
ntrials = 1000;
detector = phased.CFARDetector2D('TrainingBandSize',[4,3], ...
    'ThresholdFactor','Auto','GuardBandSize',[2,3], ...
    'ProbabilityFalseAlarm',p,'Method','SOCA','ThresholdOutputPort',true);
```

Create a 41-by-41 image containing random complex data. Then, square the data to simulate a square-law detector.

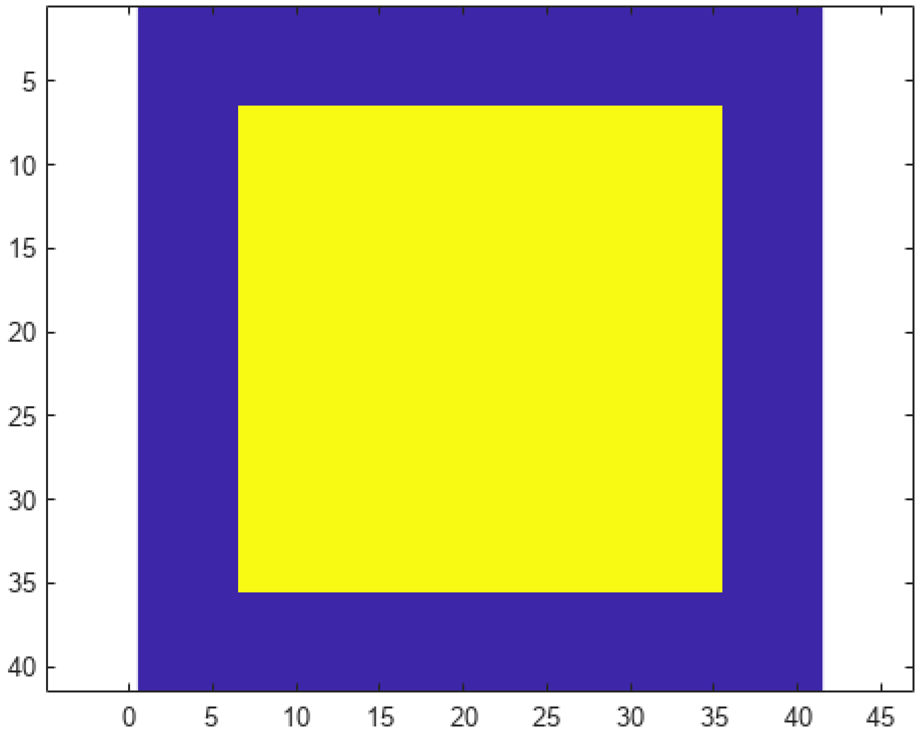
```
x = 2/sqrt(2)*(randn(rs,N,N,ntrials) + 1i*randn(rs,N,N,ntrials));
x2 = abs(x).^2;
```

Process all the cells in each image. To do this, find the row and column of each CUT cell whose training region falls entirely within each image.

```
Ngc = detector.GuardBandSize(2);
Ngr = detector.GuardBandSize(1);
Ntc = detector.TrainingBandSize(2);
Ntr = detector.TrainingBandSize(1);
cutidx = [];
colstart = Ntc + Ngc + 1;
colend = N - ( Ntc + Ngc);
rowstart = Ntr + Ngr + 1;
rowend = N - ( Ntr + Ngr);
for m = colstart:colend
    for n = rowstart:rowend
        cutidx = [cutidx,[n;m]];
    end
end
ncutcells = size(cutidx,2);
```

Display the CUT cells.

```
cutimage = zeros(N,N);
for k = 1:ncutcells
    cutimage(cutidx(1,k),cutidx(2,k)) = 1;
end
imagesc(cutimage)
axis equal
```

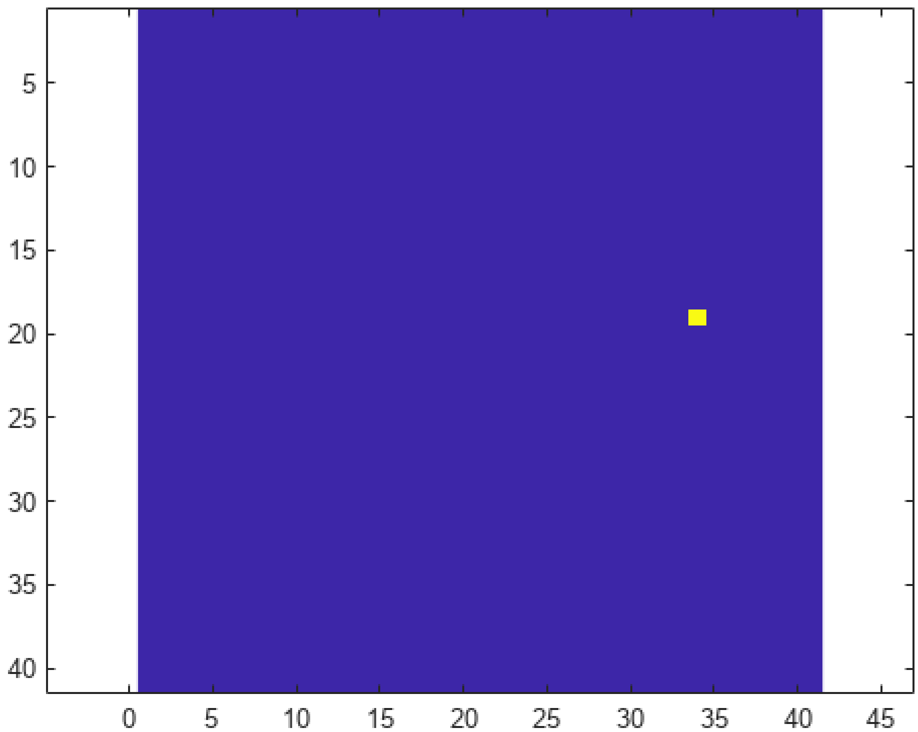


Perform the detection on all CUT cells. Return the detection classification and the threshold used to classify the cell.

```
[dets,th] = detector(x2,cutidx);
```

Find and display an image with a false alarm for illustration.

```
di = [];
for k = 1:ntrials
    d = dets(:,k);
    if (any(d) > 0)
        di = [di,k];
    end
end
idx = di(1);
detimg = zeros(N,N);
for k = 1:ncutcells
    detimg(cutidx(1,k),cutidx(2,k)) = dets(k,idx);
end
imagesc(detimg)
axis equal
```



Compute the empirical pfa.

```
pfa = sum(dets(:))/ntrials/ncutcells
```

```
pfa = 4.5898e-04
```

The empirical and specified pfa agree.

Display the average empirical threshold value over all images.

```
mean(th(:))
```

```
ans = 31.7139
```

Compute the theoretical threshold factor for the required pfa.

```
threshfactor = npwgnthresh(p,1,'noncoherent');
threshfactor = 10^(threshfactor/10);
disp(threshfactor)
```

```
    7.6009
```

The theoretical threshold factor multiplied by the noise variance should agree with the measured threshold.

```
noisevar = mean(x2(:));
disp(threshfactor*noisevar);
```

```
   30.4118
```

The theoretical threshold and empirical threshold agree to within an acceptable difference.

Detect Targets in Background Noise

Perform cell-averaging CFAR detection on a 41-by-41 matrix of cells containing five closely-spaced targets in Gaussian noise. Perform this detection on a simulation of 1000 images. Use two detectors with different guard band regions. Set the thresholds manually using the `Custom` threshold factor. Assume that the data is processed through a square law-detector and that no pulse integration is performed. Use a training cell band of 2 cells in width and 2 cells in height. For the first detector, use a guard band of 1 cell all around to separate the CUT cells from the training cells. For the second detector, use a guard band of 8 cells all around.

```
p = 5e-4;
rs = RandStream.create('mt19937ar','Seed',5);
N = 41;
ntrials = 1000;
```

Create 1000 41-by-41 images of complex random noise with standard deviation of 1.

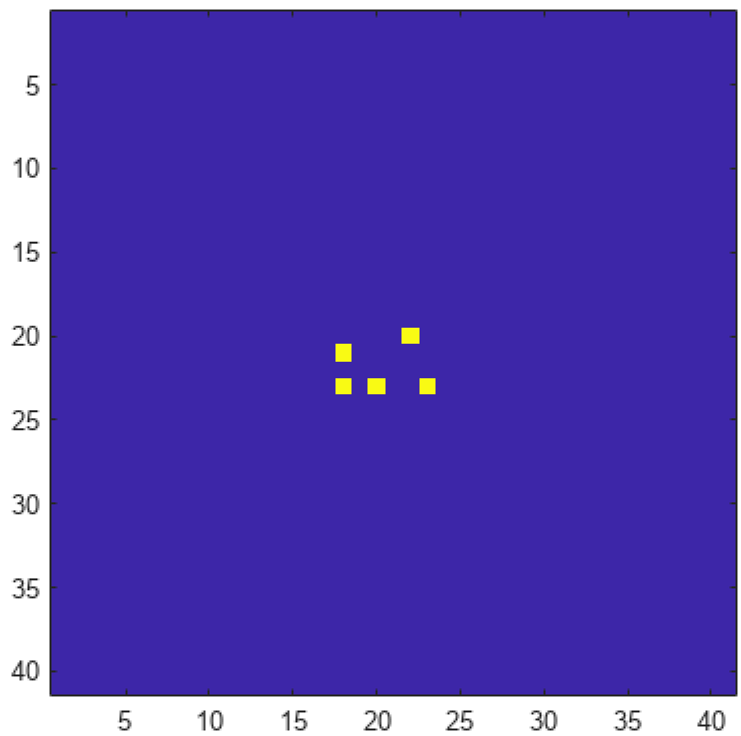
```
s = 1;
x = s/sqrt(2)*(randn(rs,N,N,ntrials) + 1i*randn(rs,N,N,ntrials));
```

Set the target cells values to 1.5. Then, square the cell values.

```
A = 1.5;
x(23,20,:) = A;
x(23,18,:) = A;
x(23,23,:) = A;
x(20,22,:) = A;
x(21,18,:) = A;
x2 = abs(x).^2;
```

Display the target cells.

```
xtgt = zeros(N,N);
xtgt(23,20,:) = A;
xtgt(23,18,:) = A;
xtgt(23,23,:) = A;
xtgt(20,22,:) = A;
xtgt(21,18,:) = A;
imagesc(xtgt)
axis equal
axis tight
```



Set the CUT cells to be the target cells.

```
cutidx(1,1) = 23;
cutidx(2,1) = 20;
cutidx(1,2) = 23;
cutidx(2,2) = 18;
cutidx(1,3) = 23;
cutidx(2,3) = 23;
cutidx(1,4) = 20;
cutidx(2,4) = 22;
cutidx(1,5) = 21;
cutidx(2,5) = 18;
```

Perform the detection on all CUT cells using two CFAR 2-D detectors. The first detector has a small guard band region. The training region can include neighboring targets which can affect the computation of the noise power. The second detector has a larger guard band region, which precludes target cells from being used in the noise computation.

Create the two CFAR detectors.

```
detector1 = phased.CFARDetector2D('TrainingBandSize',[2,2], ...
    'GuardBandSize',[1,1],'ThresholdFactor','Custom','Method','CA', ...
    'CustomThresholdFactor',2,'ThresholdOutputPort',true);
detector2 = phased.CFARDetector2D('TrainingBandSize',[2,2], ...
    'GuardBandSize',[8,8],'ThresholdFactor','Custom','Method','CA', ...
    'CustomThresholdFactor',2,'ThresholdOutputPort',true);
```

Return the detection classifications and the thresholds used to classify the cells. Then, compute the probabilities of detection.

```
[dets1,th1] = detector1(x2,cutidx);
ndets = numel(dets1(:));
pd1 = sum(dets1(:))/ndets
```

```
pd1 = 0.6416
```

```
[dets2,th2] = detector2(x2,cutidx);
pd2 = sum(dets2(:))/ndets
```

```
pd2 = 0.9396
```

The detector with the larger guard-band region has a higher pfa because the noise is more accurately estimated.

Version History

Introduced in R2016b

See Also

`phased.CFARDetector`

phased.Collector

Package: phased

Narrowband signal collector

Description

The phased.Collector System object implements a narrowband signal collector. A collector converts incident narrowband wave fields arriving from specified directions into signals to be further processed. Wave fields are incident on antenna and microphone elements, sensor arrays, or subarrays. The object collects signals in one of two ways controlled by the Wavefront property.

- If the Wavefront property is set to 'Plane', the collected signals at each element or subarray are formed from the coherent sum of all incident plane wave fields sampled at each array element or subarray.
- If the Wavefront property is set to 'Unspecified', the collected signals are formed from an independent field incident on each individual sensor element.

You can use this object to

- model arriving signals as polarized or nonpolarized fields depending upon whether the element or array supports polarization and the value of the Polarization property. Using polarization, you can receive a signal as a polarized electromagnetic field, or receive two independent signals using dual (i.e. orthogonal) polarization directions.
- model incoming acoustic fields by using nonpolarized microphone and sonar transducer array elements and by setting the “Polarization” on page 1-0 to 'None'. You must also set the PropagationSpeed to a value appropriate for the medium.
- collect fields at subarrays created by the phased.ReplicatedSubarray and phased.PartitionedArray objects. You can steer all subarrays in the same direction using the steering angle argument, STEERANG, or steer each subarray in a different direction using the subarray element weights argument, WS. You cannot set the Wavefront property to 'Unspecified' for subarrays.

To collect arriving signals at the elements or arrays:

1. Create the phased.Collector object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
collector = phased.Collector
collector = phased.Collector(Name,Value)
```

Description

`collector = phased.Collector` creates a narrowband signal collector object, `collector`, with default property values.

`collector = phased.Collector(Name,Value)` creates a narrowband signal collector with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `collector = phased.collector('Sensor',phased.URA,'OperatingFrequency',300e6)` sets the sensor array to a uniform rectangular array (URA) with default URA property values. The beamformer has an operating frequency of 300 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Sensor` — Sensor element or sensor array
`phased.ULA` array with default property values (default) | Phased Array System Toolbox sensor or array

Sensor element or sensor array, specified as a System object belonging to Phased Array System Toolbox. A sensor array can contain subarrays.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `double`

`Wavefront` — Type of incoming wavefront
`'Plane'` (default) | `'Unspecified'`

The type of incoming wavefront, specified as `'Plane'` or `'Unspecified'`:

- `'Plane'` — input signals are multiple plane waves impinging on the entire array. Each plane wave is received by all collecting elements.
- `'Unspecified'` — collected signals are independent fields incident on individual sensor elements. If the `Sensor` property is an array that contains subarrays, you cannot set the `Wavefront` property to `'Unspecified'`.

Data Types: `char`

`SensorGainMeasure` — Specify sensor gain

`'dB'` (default) | `'dBi'`

Sensor gain measure, specified as `'dB'` or `'dBi'`.

- When you set this property to `'dB'`, the input signal power is scaled by the sensor power pattern (in dB) at the corresponding direction and then combined.
- When you set this property to `'dBi'`, the input signal power is scaled by the directivity pattern (in dBi) at the corresponding direction and then combined. This option is useful when you want to compare results with the values predicted by the radar equation that uses dBi to specify the antenna gain. The computation using the `'dBi'` option is expensive as it requires an integration over all directions to compute the total radiated power of the sensor.

Dependencies

To enable this property, set the `Wavefront` property to `'Plane'`.

Data Types: `char`

`Polarization` — Polarization configuration

`'None'` (default) | `'Combined'` | `'Dual'`

Polarization configuration, specified as `'None'`, `'Combined'`, or `'Dual'`. When you set this property to `'None'`, the incident fields are considered scalar fields. When you set this property to `'Combined'`, the incident fields are polarized and represent a single arriving signal whose polarization reflects the sensor's inherent polarization. When you set this property to `'Dual'`, the *H* and *V* polarization components of the fields are independent signals.

Example: `'Dual'`

Data Types: `char`

`WeightsInputPort` — Enable weights input

`false` (default) | `true`

Enable weights input, specified as `false` or `true`. When `true`, use the object input argument `W` to specify weights. Weights are applied to individual array elements (or at the subarray level when subarrays are supported).

Data Types: `logical`

Usage

Syntax

```
Y = collector(X,ANG)
Y = collector(X,ANG,LAXES)
```

```
[YH,YV] = collector(X,ANG,LAXES)
[ ___ ] = collector( ___ ,W)
[ ___ ] = collector( ___ ,STEERANG)
[ ___ ] = collector( ___ ,WS)
```

Description

`Y = collector(X,ANG)` collects the signals, `X`, arriving from the directions specified by `ANG`. `Y` contains the collected signals.

`Y = collector(X,ANG,LAXES)` also specifies `LAXES` as the local coordinate system axes directions. To use this syntax, set the Polarization property to `'Combined'`.

`[YH,YV] = collector(X,ANG,LAXES)` returns an H-polarization component of the field, `YH`, and a V-polarization component, `YV`. To use this syntax, set the Polarization property to `'Dual'`.

`[ ___ ] = collector( ___ ,W)` also specifies `W` as array element or subarray weights. To use this syntax, set the WeightsInputPort property to `true`.

`[ ___ ] = collector( ___ ,STEERANG)` also specifies `STEERANG` as the subarray steering angle. To use this syntax, set the Sensor property to an array that supports subarrays and set the `SubarraySteering` property of that array to either `'Phase'` or `'Time'`.

`[ ___ ] = collector( ___ ,WS)` also specifies `WS` as the weights applied to each element within each subarray. To use this syntax, set the Sensor property to an array that supports subarrays and set the `SubarraySteering` of that array to `'Custom'`.

Input Arguments

X — Arriving signals

complex-valued *M*-by-*L* matrix | complex-valued 1-by-*L* cell array of structures

Arriving signals, specified as a complex-valued *M*-by-*L* matrix or complex-valued 1-by-*L* cell array of structures. *M* is the number of signal samples and *L* is the number of arrival angles. This argument represents the arriving fields.

- If the `Polarization` property value is set to `'None'`, `X` is an *M*-by-*L* matrix.
- If the `Polarization` property value is set to `'Combined'` or `'Dual'`, `X` is a 1-by-*L* cell array of structures. Each cell corresponds to a separate arriving signal. Each `struct` contains three column vectors containing the *X*, *Y*, and *Z* components of the polarized fields defined with respect to the global coordinate system.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this argument, set the `Polarization` property to `'None'` or `'Combined'`.

Data Types: `double`
Complex Number Support: Yes

ANG — Arrival directions of signals

real-valued 2-by-*L* matrix

Arrival directions of signals, specified as a real-valued 2-by-L matrix. Each column specifies an arrival direction in the form `[AzimuthAngle;ElevationAngle]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive. When the Wavefront property is `false`, the number of angles must equal the number of array elements, N. Units are in degrees.

Example: `[30,20;45,0]`

Data Types: `double`

`LAXES` — Local coordinate system
real-valued 3-by-3 orthogonal matrix

Local coordinate system, specified as a real-valued 3-by-3 orthogonal matrix. The matrix columns specify the local coordinate system's orthonormal x, y, and z axes with respect to the global coordinate system.

Example: `rotx(30)`

Dependencies

To enable this argument, set the `Polarization` property to `'Combined'` or `'Dual'`.

Data Types: `double`

`W` — Element or subarray weights
N-by-1 column vector

Element or subarray weights, specified as a complex-valued N-by-1 column vector where N is the number of array elements (or subarrays when the array supports subarrays).

Dependencies

To enable this argument, set the WeightsInputPort property to `true`.

Data Types: `double`
Complex Number Support: Yes

`WS` — Subarray element weights
complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray element weights

Sensor Array	Subarray weights
`phased.ReplicatedSubarray`	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of `WS` specifies the weights for the corresponding subarray.
`phased.PartitionedArray`	Subarrays may not have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `Sensor` property to an array that contains subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

`STEERANG` — Subarray steering angle

real-valued 2-by-1 vector

Subarray steering angle, specified as a length-2 column vector. The vector has the form `[azimuthAngle;elevationAngle]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive. Units are in degrees.

Example: `[20;15]`

Dependencies

To enable this argument, set the `Sensor` property to an array that supports subarrays and set the `SubarraySteering` property of that array to either `'Phase'` or `'Time'`

Data Types: `double`

Output Arguments

Y — Collected signal

complex-valued M-by-N matrix

Collected signal, returned as a complex-valued M-by-N matrix. M is the length of the input signal. N is the number of array elements (or subarrays when subarrays are supported). Each column corresponds to the signal collected by the corresponding array element (or corresponding subarrays when subarrays are supported).

Dependencies

To enable this argument, set the `Polarization` property to `'None'` or `'Combined'`.

Data Types: `double`

YH — Collected horizontal polarization signal
complex-valued *M*-by-*N* matrix

Collected horizontal polarization signal, returned as a complex-valued *M*-by-*N* matrix. *M* is the length of the input signal. *N* is the number of array elements (or subarrays when subarrays are supported). Each column corresponds to the signal collected by the corresponding array element (or corresponding subarrays when subarrays are supported).

Dependencies

To enable this argument, set the `Polarization` property to `'Dual'`.

Data Types: `double`

YV — Collected vertical polarization signal
complex-valued *M*-by-*N* matrix

Collected horizontal polarization signal, returned as a complex-valued *M*-by-*N* matrix. *M* is the length of the input signal. *N* is the number of array elements (or subarrays when subarrays are supported). Each column corresponds to the signal collected by the corresponding array element (or corresponding subarrays when subarrays are supported).

Dependencies

To enable this argument, set the `Polarization` property to `'Dual'`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Collect Wideband Signal at Single Antenna

Use the phased.Collector System object™ to construct a signal arriving at a single isotropic antenna from 10° azimuth and 30° elevation.

```
antenna = phased.IsotropicAntennaElement;
collector = phased.Collector('Sensor',antenna);
x = [1;0;-1];
incidentAngle = [10;30];
y = collector(x,incidentAngle)
```

```
y = 3×1

     1
     0
    -1
```

Collect Signal at Uniform Linear Array

Collect a far-field signal arriving at a 3-element uniform linear array (ULA) of isotropic antenna elements.

```
antenna = phased.ULA('NumElements',3);
collector = phased.Collector('Sensor',antenna,'OperatingFrequency',1e9);
x = [1;0;-1];
incidentAngle = [10 30]';
y = collector(x,incidentAngle)
```

```
y = 3×3 complex

  -0.0051 - 1.0000i   1.0000 + 0.0000i  -0.0051 + 1.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0051 + 1.0000i  -1.0000 + 0.0000i   0.0051 - 1.0000i
```

Collect Different Signals at Array Elements

Collect different signals at a three-element array. Each input signal comes from a different direction.

```
array = phased.ULA('NumElements',3);
collector = phased.Collector('Sensor',array,'OperatingFrequency',1e9,...
    'Wavefront','Unspecified');
```

Each column is a signal for one element.

```
x = rand(10,3)
```

```
x = 10×3

    0.8147    0.1576    0.6557
    0.9058    0.9706    0.0357
    0.1270    0.9572    0.8491
```

```
    0.9134    0.4854    0.9340
    0.6324    0.8003    0.6787
    0.0975    0.1419    0.7577
    0.2785    0.4218    0.7431
    0.5469    0.9157    0.3922
    0.9575    0.7922    0.6555
    0.9649    0.9595    0.1712
```

Specify three incident angles.

```
incidentAngles = [10 0; 20 5; 45 2]';
y = collector(x,incidentAngles)
```

```
y = 10×3

    0.8147    0.1576    0.6557
    0.9058    0.9706    0.0357
    0.1270    0.9572    0.8491
    0.9134    0.4854    0.9340
    0.6324    0.8003    0.6787
    0.0975    0.1419    0.7577
    0.2785    0.4218    0.7431
    0.5469    0.9157    0.3922
    0.9575    0.7922    0.6555
    0.9649    0.9595    0.1712
```

Collect Plane Wave at ULA

Construct a 4-element uniform linear array (ULA). The array operating frequency is 1 GHz. The array element spacing is one half the corresponding wavelength. Model the collection of a 200 Hz sinusoid from the far field incident on the array at 45° azimuth and 10° elevation.

Create the array.

```
fc = 1e9;
lambda = physconst('LightSpeed')/fc;
array = phased.ULA('NumElements',4,'ElementSpacing',lambda/2);
```

Create the sinusoid signal.

```
t = linspace(0,1,1e3);
x = cos(2*pi*200*t)';
```

Construct the collector object and obtain the received signal.

```
collector = phased.Collector('Sensor',array, ...
    'PropagationSpeed',physconst('LightSpeed'),'Wavefront','Plane', ...
    'OperatingFrequency',fc);
incidentangle = [45;10];
receivedsig = collector(x,incidentangle);
```

Measure Target Scattering Matrix Using Dual Polarization

Use a dual-polarization system to obtain target scattering information. Simulate a transmitter and receiver where the vertical and horizontal components are transmitted successively using the input ports of the transmitter. The signals from the two polarization output ports of the receiver are then used to determine the target scattering matrix.

```
scmat = [0 1i; 1i 2];
radiator = phased.Radiator('Sensor', ...
    phased.CustomAntennaElement('SpecifyPolarizationPattern',true), ...
    'Polarization','Dual');
target = phased.RadarTarget('EnablePolarization',true,'ScatteringMatrix', ...
    scmat);
collector = phased.Collector('Sensor', ...
    phased.CustomAntennaElement('SpecifyPolarizationPattern',true), ...
    'Polarization','Dual');
xh = 1;
xv = 1;
```

Transmit a horizontal component and display the reflected Shh and Svh polarization components.

```
x = radiator(xh,0,[0;0],eye(3));
xrefl = target(x,[0;0],eye(3));
[Shh,Svh] = collector(xrefl,[0;0],eye(3))
```

```
Shh = 0

Svh = 0.0000 + 3.5474i
```

Transmit a vertical component and display the reflected Shv and Svv polarization components.

```
x = radiator(0,xv,[0;0],eye(3));
xrefl = target(x,[0;0],eye(3));
[Shv,Svv] = collector(xrefl,[0;0],eye(3))
```

```
Shv = 0.0000 + 3.5474i

Svv = 7.0947
```

Algorithms

If the `Wavefront` property value is `'Plane'`, `phased.Collector` collects each plane wave signal using the phase approximation of the time delays across collecting elements in the far field.

If the `Wavefront` property value is `'Unspecified'`, `phased.Collector` collects each channel independently.

For further details, see [1].

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.WidebandCollector` | `phased.Radiator` | `phased.WidebandRadiator`

step

System object: `phased.Collector`
Package: `phased`

Collect signals

Syntax

```
Y = step(H,X,ANG)
Y = step(H,X,ANG,LAXES)
Y = step(H,X,ANG,WEIGHTS)
Y = step(H,X,ANG,STEERANGLE)
Y = step(H,X,ANG,LAXES,WEIGHTS,STEERANGLE)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X,ANG)` collects signals `X` arriving from directions `ANG`. The collection process depends on the `Wavefront` property of `H`, as follows:

- If `Wavefront` has the value `'Plane'`, each collecting element collects all the far field signals in `X`. Each column of `Y` contains the output of the corresponding element in response to all the signals in `X`.
- If `Wavefront` has the value `'Unspecified'`, each collecting element collects only one impinging signal from `X`. Each column of `Y` contains the output of the corresponding element in response to the corresponding column of `X`. The `'Unspecified'` option is available when the `Sensor` property of `H` does not contain subarrays.

`Y = step(H,X,ANG,LAXES)` uses `LAXES` as the local coordinate system axes directions. This syntax is available when you set the `EnablePolarization` property to `true`.

`Y = step(H,X,ANG,WEIGHTS)` uses `WEIGHTS` as the weight vector. This syntax is available when you set the `WeightsInputPort` property to `true`.

`Y = step(H,X,ANG,STEERANGLE)` uses `STEERANGLE` as the subarray steering angle. This syntax is available when you configure `H` so that `H.Sensor` is an array that contains subarrays and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

`Y = step(H,X,ANG,LAXES,WEIGHTS,STEERANGLE)` combines all input arguments. This syntax is available when you configure `H` so that `H.WeightsInputPort` is `true`, `H.Sensor` is an array that contains subarrays, and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of

the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Collector object.

X

Arriving signals. Each column of `X` represents a separate signal. The specific interpretation of `X` depends on the `Wavefront` property of `H`.

Wavefront Property Value	Description
`'Plane'`	Each column of `X` is a far field signal.
`'Unspecified'`	Each column of `X` is the signal impinging on the corresponding element. In this case, the number of columns in `X` must equal the number of collecting elements in the `Sensor` property.

- If the `EnablePolarization` property value is set to `false`, `X` is a matrix. The number of columns of the matrix equals the number of separate signals.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
- If the `EnablePolarization` property value is set to `true`, `X` is a row vector of MATLAB `struct` type. The dimension of the `struct` array equals the number of separate signals. Each `struct` member contains three column-vector fields, `X`, `Y`, and `Z`, representing the *x*, *y*, and *z* components of the polarized wave vector signals in the global coordinate system.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

ANG

Incident directions of signals, specified as a two-row matrix. Each column specifies the incident direction of the corresponding column of `X`. Each column of `ANG` has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

LAXES

Local coordinate system. `LAXES` is a 3-by-3 matrix whose columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively. Each axis is specified in terms of `[x;y;z]` with respect to the global coordinate system. This argument is only used when the `EnablePolarization` property is set to `true`.

WEIGHTS

Vector of weights. WEIGHTS is a column vector of length M, where M is the number of collecting elements.

Default: ones(M,1)

STEERANGLE

Subarray steering angle, specified as a length-2 column vector. The vector has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

Output Arguments

Y

Collected signals. Each column of Y contains the output of the corresponding element. The output is the response to all the signals in X, or one signal in X, depending on the Wavefront property of H.

Examples

Collect Plane Wave at ULA

Construct a 4-element uniform linear array (ULA). The array operating frequency is 1 GHz. The array element spacing is one half the corresponding wavelength. Model the collection of a 200 Hz sinusoid from the far field incident on the array at 45° azimuth and 10° elevation.

Create the array.

```
fc = 1e9;
lambda = physconst('LightSpeed')/fc;
array = phased.ULA('NumElements',4,'ElementSpacing',lambda/2);
```

Create the sinusoid signal.

```
t = linspace(0,1,1e3);
x = cos(2*pi*200*t)';
```

Construct the collector object and obtain the received signal.

```
collector = phased.Collector('Sensor',array, ...
    'PropagationSpeed',physconst('LightSpeed'),'Wavefront','Plane', ...
    'OperatingFrequency',fc);
incidentangle = [45;10];
receivedsig = collector(x,incidentangle);
```

Algorithms

If the Wavefront property value is 'Plane', phased.Collector collects each plane wave signal using the phase approximation of the time delays across collecting elements in the far field.

If the `Wavefront` property value is `'Unspecified'`, `phased.Collector` collects each channel independently.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

phased.ConformalArray

Package: phased

Conformal array

Description

The ConformalArray object constructs a conformal array. A conformal array can have elements in any position pointing in any direction.

To compute the response for each element in the array for specified directions:

1. Define and set up your conformal array. See "Construction" on page 1-156.
2. Call step to compute the response according to the properties of phased.ConformalArray. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.ConformalArray creates a conformal array System object, H. The object models a conformal array formed with identical sensor elements.

H = phased.ConformalArray(Name,Value) creates object, H, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

H = phased.ConformalArray(POS,NV,Name,Value) creates a conformal array object, H, with the ElementPosition property set to POS, the ElementNormal property set to NV, and other specified property Names set to the specified Values. POS and NV are value-only arguments. When specifying a value-only argument, specify all preceding value-only arguments. You can specify name-value arguments in any order.

Properties

Element

Element of array

Sensor element array, specified as a Phased Array System Toolbox antenna, microphone, or transducer element or an Antenna Toolbox™ antenna System object

Default: isotropic antenna element System object with default properties

ElementPosition

Element positions

`ElementPosition` specifies the positions of the elements in the conformal array. `ElementPosition` must be a 3-by-N matrix, where N indicates the number of elements in the conformal array. Each column of `ElementPosition` represents the position, in the form `[x; y; z]` (in meters), of a single element in the local coordinate system of the array. The local coordinate system has its origin at an arbitrary point. The default value of this property represents a single element at the origin of the local coordinate system.

Default: `[0; 0; 0]`

`ElementNormal`

Element normal directions

`ElementNormal` specifies the normal directions of the elements in the conformal array. Angle units are degrees. The value assigned to `ElementNormal` must be either a 2-by-*N* matrix or a 2-by-1 column vector. The variable *N* indicates the number of elements in the array. If the value of `ElementNormal` is a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the value of `ElementNormal` is a 2-by-1 column vector, it specifies the same pointing direction for all elements in the array.

You can use the `ElementPosition` and `ElementNormal` properties to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Default: `[0; 0]`

`Taper`

Element taper or weighting

Element tapering or weighting, specified as a complex-valued scalar, 1-by-*N* row vector, or *N*-by-1 column vector. Weights are applied to each element in the sensor array. *N* is the number of elements along in the array as determined by the size of the `ElementPosition` property. If the `Taper` parameter is a scalar, the same taper value is applied to all elements. If the value of `Taper` is a vector, each taper values is applied to the corresponding element.

Default: 1

Methods

Specific to `phased.ConformalArray` Object	
`beamwidth`	Compute and display beamwidth of an array
`collectPlaneWave`	Simulate received plane waves
`directivity`	Directivity of conformal array
`getElementNormal`	Normal vector to array elements

Specific to phased.ConformalArray Object	
getElementPosition	Positions of array elements
getNumElements	Number of elements in array
getTaper	Array element tapers
isPolarizationCapable	Polarization capability
pattern	Plot conformal array pattern
patternAzimuth	Plot conformal array directivity or pattern versus azimuth
patternElevation	Plot conformal array directivity or pattern versus elevation
perturbations	Perturbations defined on phased array
perturbedArray	Apply perturbations to phased array
perturbedPattern	Compute and plot azimuth pattern of perturbed array
step	Output responses of array elements
viewArray	View array geometry

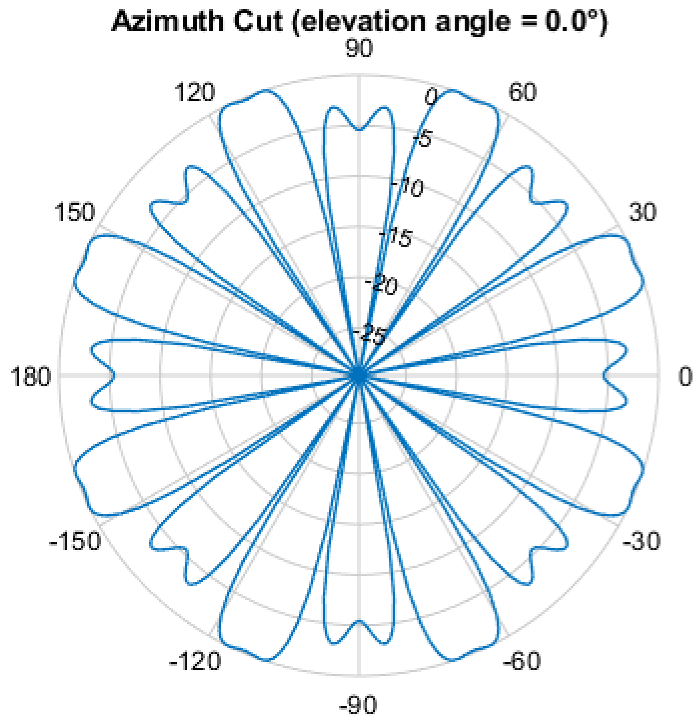
Common to All System Objects	
release	Allow System object property value changes

Examples

Plot Power Pattern of 8-Element Uniform Circular Array

Using the ConformalArray System object™, construct an 8-element uniform circular array (UCA) of isotropic antenna elements. Plot a normalized azimuth power pattern at 0 degrees elevation. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
N = 8;
azang = (0:N-1)*360/N-180;
sCA = phased.ConformalArray(...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
fc = 1e9;
c = physconst('LightSpeed');
pattern(sCA,fc,[-180:180],0,...
    'PropagationSpeed',c,'Type','powerdb',...
    'CoordinateSystem','polar')
```



Plot Pattern of 31-Element Uniform Circular Sonar Array

Construct a 31-element acoustic uniform circular sonar array (UCA) using the ConformalArray System object™. Assume the array is one meter in diameter. Using the `ElevationAngles` parameter, restrict the display to +/-40 degrees in 0.1 degree increments. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s.

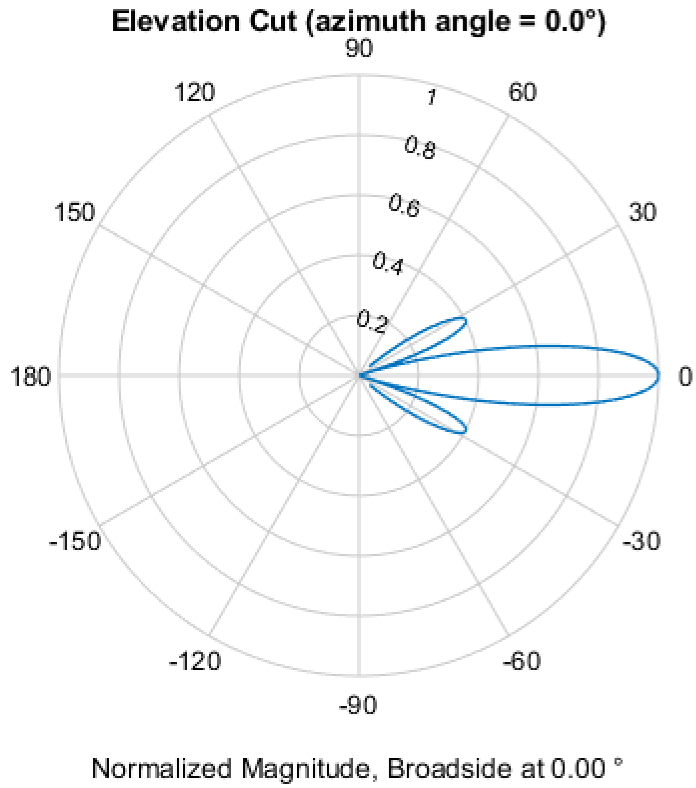
Construct the array

```
N = 31;
theta = (0:N-1)*360/N-180;
Radius = 0.5;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[0,10000],'BackBaffled',true);
sArray = phased.ConformalArray('Element',sMic,...
    'ElementPosition',Radius*[zeros(1,N);cosd(theta);sind(theta)],...
    'ElementNormal',[ones(1,N);zeros(1,N)]);
```

Plot the magnitude pattern

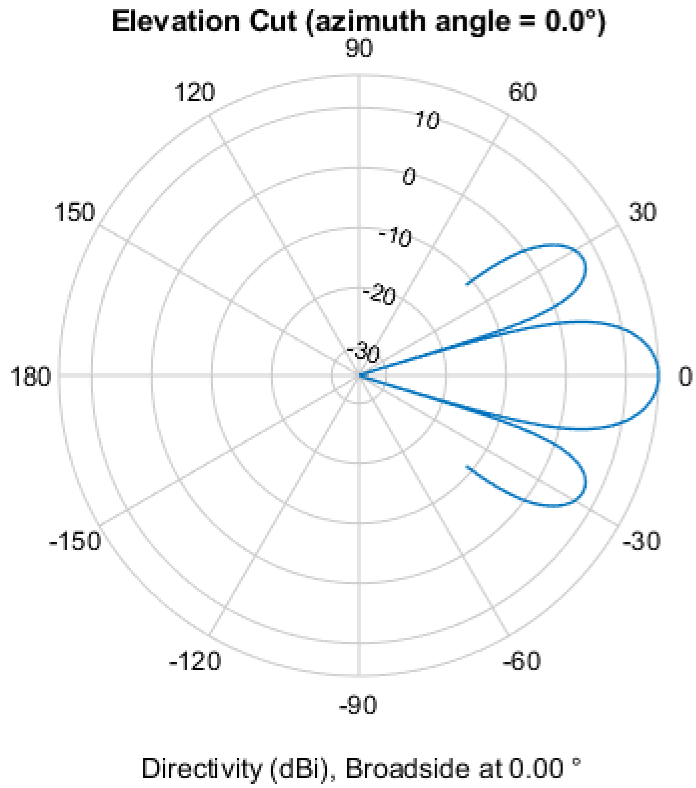
```
fc = 4000;
c = 1500.0;
pattern(sArray,fc,0,[-40:0.1:40],...
    'PropagationSpeed',c,...
```

```
    'CoordinateSystem','polar',...
    'Type','efield')
```



Plot the directivity pattern

```
pattern(sArray,fc,0,[-40:0.1:40],...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','directivity')
```



Version History

Introduced in R2011a

References

[1] Josefsson, L. and P. Persson. *Conformal Array Antenna Theory and Design*. Piscataway, NJ: IEEE Press, 2006.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, `plotResponse`, and `viewArray` methods are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

phased.ReplicatedSubarray | phased.PartitionedArray | phased.CosineAntennaElement | phased.CustomAntennaElement | phased.IsotropicAntennaElement | phased.ULA | phased.URA | uv2azel | phased.UCA | phitheta2azel

Topics

Phased Array Gallery

directivity

System object: `phased.ConformalArray`
Package: `phased`

Directivity of conformal array

Syntax

```
D = directivity(H,FREQ,ANGLE)
D = directivity(H,FREQ,ANGLE,Name,Value)
```

Description

`D = directivity(H,FREQ,ANGLE)` computes the “Directivity” on page 1-166 of a conformal array of antenna or microphone elements, `H`, at frequencies specified by the `FREQ` and in angles of direction specified by the `ANGLE`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`D = directivity(H,FREQ,ANGLE,Name,Value)` computes the directivity with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

H — Conformal array

System object

Conformal array specified as a `phased.ConformalArray` System object.

Example: `H = phased.ConformalArray;`

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

ANGLE — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

PropagationSpeed — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Conformal Array

Compute the directivity of a circular array constructed using a conformal array System object™.

Construct a 21-element uniform circular sonar array (UCA) of backbaffled omnidirectional microphones. The array is one meter in diameter. Set the operating frequency to 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s.

```
N = 21;
theta = (0:N-1)*360/N-180;
Radius = 0.5;
myMic = phased.OmnidirectionalMicrophoneElement;
myMicFrequencyRange = [0,5000];
myMic.BackBaffled = true;
myArray = phased.ConformalArray;
myArray.Element = myMic;
myArray.ElementPosition = Radius*[zeros(1,N);cosd(theta);sind(theta)];
myArray.ElementNormal = [ones(1,N);zeros(1,N)];
c = 1500.0;
fc = 4000;
```

Steer the array to 30 degrees in azimuth and compute the directivity in the steering direction.

```
lambda = c/fc;
ang = [30;0];
w = steervec(getElementPosition(myArray)/lambda,ang);
d = directivity(myArray,fc,ang,...
    'PropagationSpeed',c,...
    'Weights',w)
```

```
d = 15.1633
```

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternAzimuth` | `patternElevation`

collectPlaneWave

System object: `phased.ConformalArray`
Package: `phased`

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

`Y = collectPlaneWave(H,X,ANG)` returns the received signals at the sensor array, `H`, when the input signals indicated by `X` arrive at the array from the directions specified in `ANG`.

`Y = collectPlaneWave(H,X,ANG,FREQ)`, in addition, specifies the incoming signal carrier frequency in `FREQ`.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)`, in addition, specifies the signal propagation speed in `C`.

Input Arguments

H

Array object.

X

Incoming signals, specified as an M-column matrix. Each column of `X` represents an individual incoming signal.

ANG

Directions from which incoming signals arrive, in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column specifies the direction of arrival of the corresponding signal in `X`. Each column of `ANG` is in the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `ANG` is a row vector of length M, each entry in `ANG` specifies the azimuth angle. In this case, the corresponding elevation angle is assumed to be 0°.

FREQ

Carrier frequency of signal in hertz. `FREQ` must be a scalar.

Default: `3e8`

C

Propagation speed of signal in meters per second.

Default: Speed of light

Output Arguments

Y

Received signals. Y is an N-column matrix, where N is the number of elements in the array H. Each column of Y is the received signal at the corresponding array element, with all incoming signals combined.

Examples

Simulate Received Signals at Conformal Array

Simulate two received signals at an 8-element uniform circular array. The signals arrive from 10° and 30° azimuth, respectively. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
N = 8;
azang = (0:N-1)*360/N-180;
array = phased.ConformalArray('ElementPosition', ...
    [cosd(azang);sind(azang);zeros(1,N)],'ElementNormal',[azang;zeros(1,N)]);
y = collectPlaneWave(array,randn(4,2),[10 30],100e6);
```

Algorithms

`collectPlaneWave` modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. The method does not account for the response of individual elements in the array.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

getElementNormal

System object: phased.ConformalArray
Package: phased

Normal vector to array elements

Syntax

```
normvec = getElementNormal(sConfArray)
normvec = getElementNormal(sConfArray,elemidx)
```

Description

normvec = getElementNormal(sConfArray) returns the normal vectors of the array elements of the phased.sConfArray System object, sConfArray. The output argument normvec is a 2-by-*N* matrix, where *N* is the number of elements in array, sConfArray. Each column of normvec defines the normal direction of an element in the local coordinate system in the form[az;el]. Units are degrees. The origin of the local coordinate system is defined by the phase center of the array.

normvec = getElementNormal(sConfArray,elemidx) returns only the normal vectors of the elements specified in the element index vector, elemidx. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

sConfArray — Conformal array
phased.ConformalArray System object

Conformal array, specified as a phased.ConformalArray System object.

Example: phased.ConformalArray

elemidx — Element indices
all array elements (default) | integer-valued 1-by-*M* row vector | integer-valued *M*-by-1 column vector

Element indices , specified as a 1-by-*M* or *M*-by-1 vector. Index values lie in the range 1 to *N* where *N* is the number of elements of the array. When elemidx is specified, getElementNormal returns the normal vectors of the elements contained in elemidx.

Example: [1,5,4]

Output Arguments

normvec — Element normal vectors
2-by-*P* real-valued vector

Element normal vectors, specified as a 2-by-*P* real-valued vector. Each column of normvec takes the form [az,el]. When elemidx is not specified, *P* equals the array dimension. When elemidx is specified, *P* equals the length of elemidx, *M*.

Examples

Conformal Array Element Normals

Construct a 5-element acoustic cross array (UCA) using the ConformalArray System object™. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s. Display the array normal vectors.

```
N = 5;
fc = 4000;
c = 1500.0;
lam = c/fc;
x = zeros(1,N);
y = [-1,0,1,0,0]*lam/2;
z = [0,0,0,-1,1]*lam/2;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[0,10000],'BackBaffled',true);
sConformArray = phased.ConformalArray('Element',sMic,...
    'ElementPosition',[x;y;z],...
    'ElementNormal',[45*ones(1,N);zeros(1,N)]);
pos = getElementPosition(sConformArray)
```

```
pos = 3×5

         0         0         0         0         0
   -0.1875         0    0.1875         0         0
         0         0         0   -0.1875    0.1875
```

```
normvec = getElementNormal(sConformArray)
```

```
normvec = 2×5

    45    45    45    45    45
     0     0     0     0     0
```

Version History

Introduced in R2016a

getElementPosition

System object: `phased.ConformalArray`
Package: `phased`

Positions of array elements

Syntax

```
POS = getElementPosition(H)
POS = getElementPosition(H,ELEIDX)
```

Description

`POS = getElementPosition(H)` returns the element positions of the conformal array `H`. `POS` is an 3xN matrix where N is the number of elements in `H`. Each column of `POS` defines the position of an element in the local coordinate system, in meters, using the form [x; y; z].

For details regarding the local coordinate system of the conformal array, enter `phased.ConformalArray.coordinateSystemInfo`.

`POS = getElementPosition(H,ELEIDX)` returns the positions of the elements that are specified in the element index vector `ELEIDX`.

Examples

Element Positions of Conformal Array

Construct a three-element conformal array and obtain the element positions.

```
array = phased.ConformalArray('ElementPosition',[-1,0,1;0,0,0;0,0,0]);
pos = getElementPosition(array)
```

```
pos = 3×3

    -1     0     1
     0     0     0
     0     0     0
```

getNumElements

System object: `phased.ConformalArray`
Package: `phased`

Number of elements in array

Syntax

`N = getNumElements(H)`

Description

`N = getNumElements(H)` returns the number of elements, N, in the conformal array object `H`.

Examples

Number of Elements of Conformal Array

Construct a three-element conformal array and obtain the number of elements.

```
array = phased.ConformalArray('ElementPosition',[-1,0,1;0,0,0;0,0,0]);
N = getNumElements(array)
```

```
N = 3
```

getTaper

System object: `phased.ConformalArray`
Package: `phased`

Array element tapers

Syntax

`wts = getTaper(h)`

Description

`wts = getTaper(h)` returns the tapers applied to each element of a conformal array, `h`. Tapers are often referred to as weights.

Input Arguments

`h` — Conformal array

`phased.ConformalArray` System object

Conformal array specified as a `phased.ConformalArray` System object.

Output Arguments

`wts` — Array element tapers

N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1, complex-valued vector, where *N* is the number of elements in the array.

Examples

Create and View a Tapered Array

Create a two-ring tapered disk array

Create a two-ring disk array and set the taper values on the outer ring to be smaller than those on the inner ring.

```
elemAngles = ([0:5]*360/6);
elemPosInner = 0.5*[zeros(size(elemAngles));...
    cosd(elemAngles);...
    sind(elemAngles)];
elemPosOuter = [zeros(size(elemAngles));...
    cosd(elemAngles);...
    sind(elemAngles)];
elemNorms = repmat([0;0],1,12);
taper =  [ones(size(elemAngles)),0.3*ones(size(elemAngles))];
```

```
ha = phased.ConformalArray(...
    [elemPosInner,elemPosOuter],elemNorms,'Taper',taper);
```

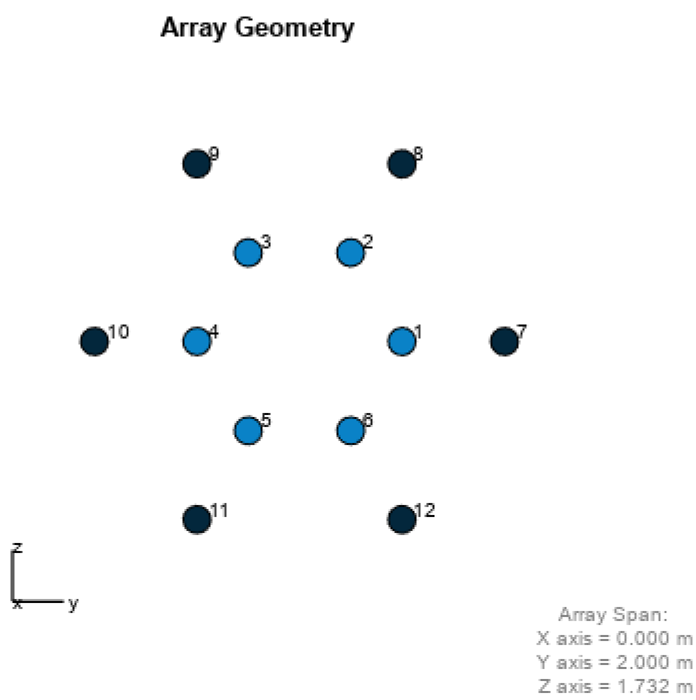
Display the taper values

```
w = getTaper(ha)
```

```
w = 12×1

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    0.3000
    0.3000
    0.3000
    0.3000
      ⋮
```

View the array

```
viewArray(ha,'ShowTaper',true,'ShowIndex','all');
```



isPolarizationCapable

System object: phased.ConformalArray
Package: phased

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h — Conformal array

Conformal array specified as a `phased.ConformalArray` System object.

Output Arguments

`flag` — Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

Conformal Array of Short-Dipole Antennas Supports Polarization

Show that a circular conformal array of `phased.ShortDipoleAntennaElement` antenna elements supports polarization.

```
N = 8;
azang = (0:N-1)*360/N-180;
antenna = phased.ShortDipoleAntennaElement;
array = phased.ConformalArray(...
    'Element',antenna,'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

The returned value `1` shows that this array supports polarization.

pattern

System object: `phased.ConformalArray`
Package: `phased`

Plot conformal array pattern

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sArray,FREQ)` plots the 3-D array directivity pattern (in dBi) for the array specified in `sArray`. The operating frequency is specified in `FREQ`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`pattern(sArray,FREQ,AZ)` plots the array directivity pattern at the specified azimuth angle.

`pattern(sArray,FREQ,AZ,EL)` plots the array directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the array pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See “Convert plotResponse to pattern” on page 1-184 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sArray` — Conformal array

System object

Conformal array, specified as a `phased.ConformalArray` System object.

Example: `sArray= phased.ConformalArray;`

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

CoordinateSystem — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Orientation` — Array orientation
`[0;0;0]`. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

`Normalize` — Display normalize pattern
`true` (default) | `false`

Display normalized pattern, specified as the comma-separated pair consisting of `'Normalize`' and a Boolean. Set this parameter to `true` to display a normalized pattern. This parameter does not apply when you set `'Type'` to `'directivity'`. Directivity patterns are already normalized.

Data Types: `logical`

`ShowArray` — View array geometry
`false` (default) | `true`

View the array geometry along with the 3D radiation pattern, specified as `false` or `true`.

Data Types: `logical`

`ShowLocalCoordinates` — Show local coordinate axes
`true` (default) | `false`

Show the local coordinate axes, specified as `true` or `false`.

Data Types: `logical`

`ShowColorbar` — Show colorbar
`true` (default) | `false`

Show the colorbar, specified as `true` or `false`.

Data Types: `logical`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights`' and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`PAT` — Array pattern

M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of `PAT` correspond to the dimensions of the output arguments `AZ_ANG` and `EL_ANG`.

`AZ_ANG` — Azimuth angles

scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in `AZ`. The columns of `PAT` correspond to the values in `AZ_ANG`. Units are in degrees.

`EL_ANG` — Elevation angles

scalar | 1-by-*M* real-valued row vector

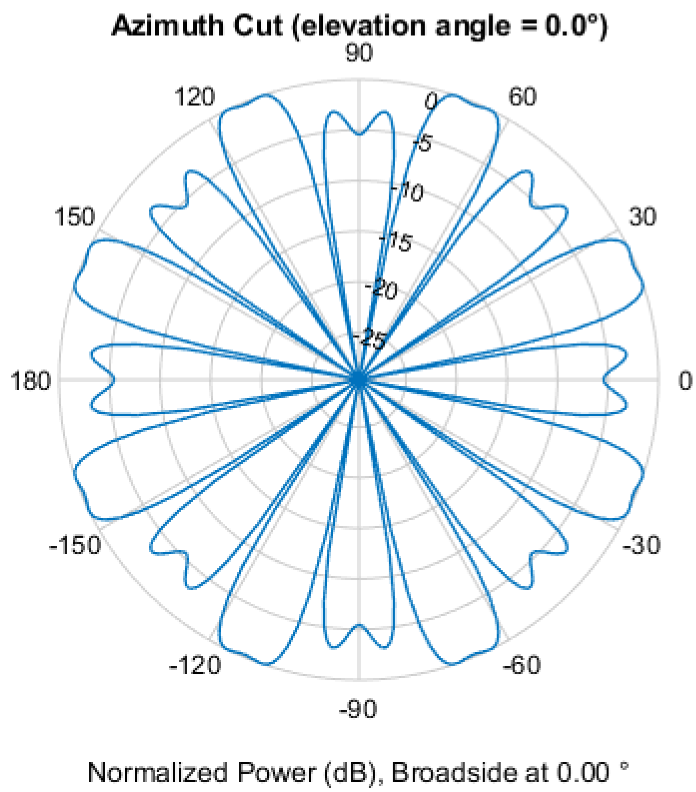
Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

Examples

Plot Power Pattern of 8-Element Uniform Circular Array

Using the ConformalArray System object™, construct an 8-element uniform circular array (UCA) of isotropic antenna elements. Plot a normalized azimuth power pattern at 0 degrees elevation. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
N = 8;
azang = (0:N-1)*360/N-180;
sCA = phased.ConformalArray(...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
fc = 1e9;
c = physconst('LightSpeed');
pattern(sCA,fc,[-180:180],0,...
    'PropagationSpeed',c,'Type','powerdb',...
    'CoordinateSystem','polar')
```



Plot Pattern of 31-Element Uniform Circular Sonar Array

Construct a 31-element acoustic uniform circular sonar array (UCA) using the ConformalArray System object™. Assume the array is one meter in diameter. Using the `ElevationAngles` parameter, restrict the display to +/-40 degrees in 0.1 degree increments. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s.

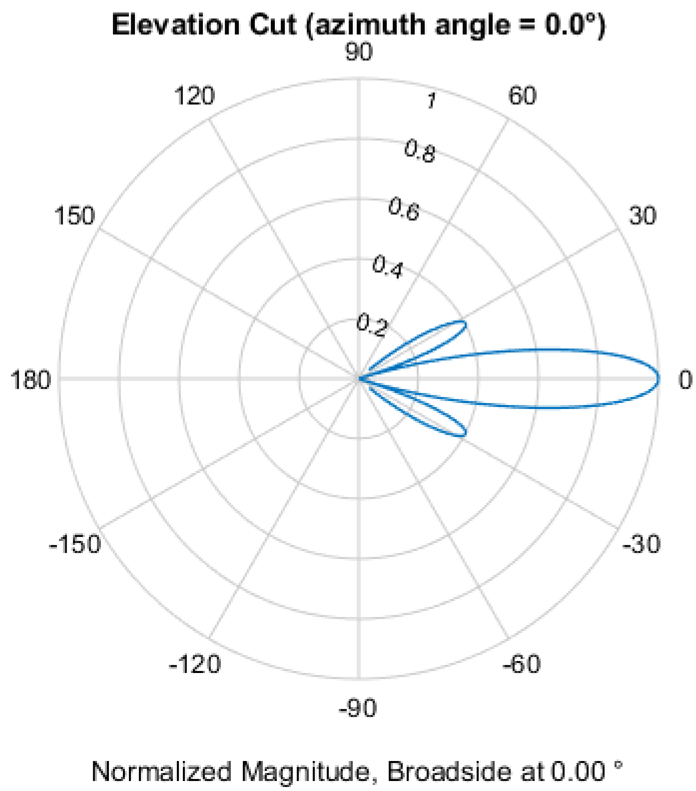
Construct the array

```
N = 31;
theta = (0:N-1)*360/N-180;
Radius = 0.5;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[0,10000],'BackBaffled',true);
```

```
sArray = phased.ConformalArray('Element',sMic,...
    'ElementPosition',Radius*[zeros(1,N);cosd(theta);sind(theta)],...
    'ElementNormal',[ones(1,N);zeros(1,N)]);
```

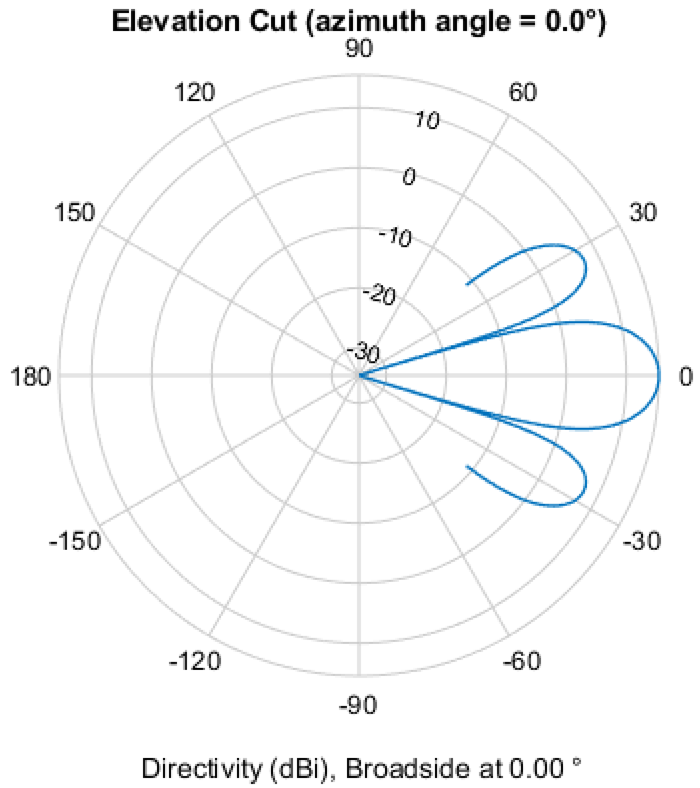
Plot the magnitude pattern

```
fc = 4000;
c = 1500.0;
pattern(sArray,fc,0,[-40:0.1:40],...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','efield')
```



Plot the directivity pattern

```
pattern(sArray,fc,0,[-40:0.1:40],...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','directivity')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on "Element Directivity" and "Array Directivity".

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>`'Format'` and `'RespCut'` name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using `plotResponse`.

<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set `'RespCut'` to `'Az'` or `'El'`. Set `'Format'` to `'line'` or `'polar'`.

Set the display axis using either the `'AzimuthAngles'` or `'ElevationAngles'` name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set `'RespCut'` to `'3D'`. Set `'Format'` to `'line'` or `'polar'`.

Set the display axis using both the `'AzimuthAngles'` and`'ElevationAngles'` name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set `'RespCut'` to`'U'`. Set `'Format'` to `'UV'`. Set the display range using the `'UGrid'` name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set `'RespCut'` to`'3D'`. Set `'Format'` to</td></tr>
</table>
</td>
<td>`'CoordinateSystem'` name-value pair used together with the `AZ` and `EL` input arguments.

`'CoordinateSystem'` has the same options as the `plotResponse` method `'Format'`name-value pair, except that `'line'` is now named `'rectangular'`. The table shows how to create different types of plots using `pattern`.

<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set `'CoordinateSystem'` to `'rectangular'` or `'polar'`. Specify either `AZ` or `EL` as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set `'CoordinateSystem'` to `'rectangular'` or `'polar'`. Specify both `AZ` and `EL` as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set `'CoordinateSystem'` to `'uv'`. Use `AZ` to specify a U-space vector. Use `EL` to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set `'CoordinateSystem'` to `'uv'`. Use `AZ` to specify a U-space vector. Use `EL` to specify a V-space vector.</td></tr>
</table>
If you set `CoordinateSystem` to `'uv'`, enter the UV grid values using `AZ` and `EL`.</td>
</tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
	Display space | | 'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.	
'CutAngle' name-value pair	Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.	No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.
'NormalizeResponse' name-value pair	Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.	Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.
'OverlayFreq' name-value pair	Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.	'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot. The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.
'Polarization' name-value pair	Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.	'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.
'Unit' name-value pair	Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.	'Type' name-value pair, uses equivalent options with different names **plotResponse** → **pattern** 'db' → 'powerdb' 'mag' → 'efield' 'pow' → 'power' 'dbi' → 'directivity'
'Weights' name-value pair	Array element tapers (or weights).	'Weights' name-value pair (no change).
'AzimuthAngles' name-value pair	Azimuth angles used to display the antenna or array response.	AZ argument

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `patternElevation`

patternAzimuth

System object: `phased.ConformalArray`
Package: `phased`

Plot conformal array directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sArray,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(sArray,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sArray,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sArray` — Conformal array

System object

Conformal array, specified as a `phased.ConformalArray` System object.

Example: `sArray= phased.ConformalArray;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

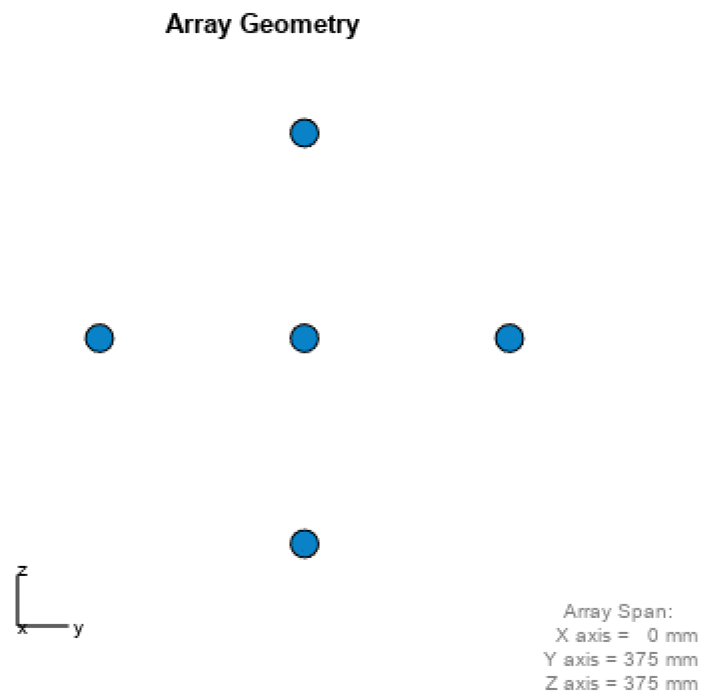
Examples

Plot Azimuth Pattern of 5-Element Cross Sonar Array

Construct a 5-element acoustic cross array (UCA) using the ConformalArray System object™. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s. Plot the array patterns at two different elevation angles.

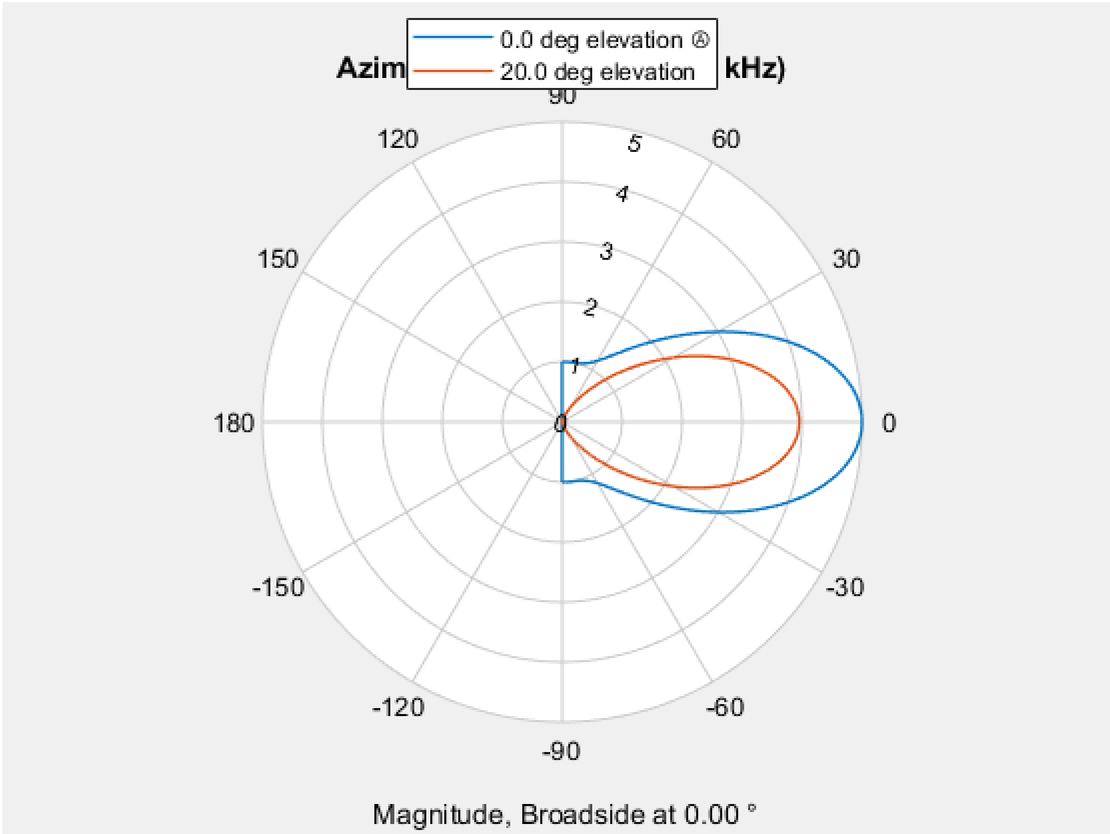
Construct and view array

```
N = 5;
fc = 4000;
c = 1500.0;
lam = c/fc;
x = zeros(1,N);
y = [-1,0,1,0,0]*lam/2;
z = [0,0,0,-1,1]*lam/2;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[0,10000],'BackBaffled',true);
sArray = phased.ConformalArray('Element',sMic,...
    'ElementPosition',[x;y;z],...
    'ElementNormal',[zeros(1,N);zeros(1,N)]);
viewArray(sArray)
```



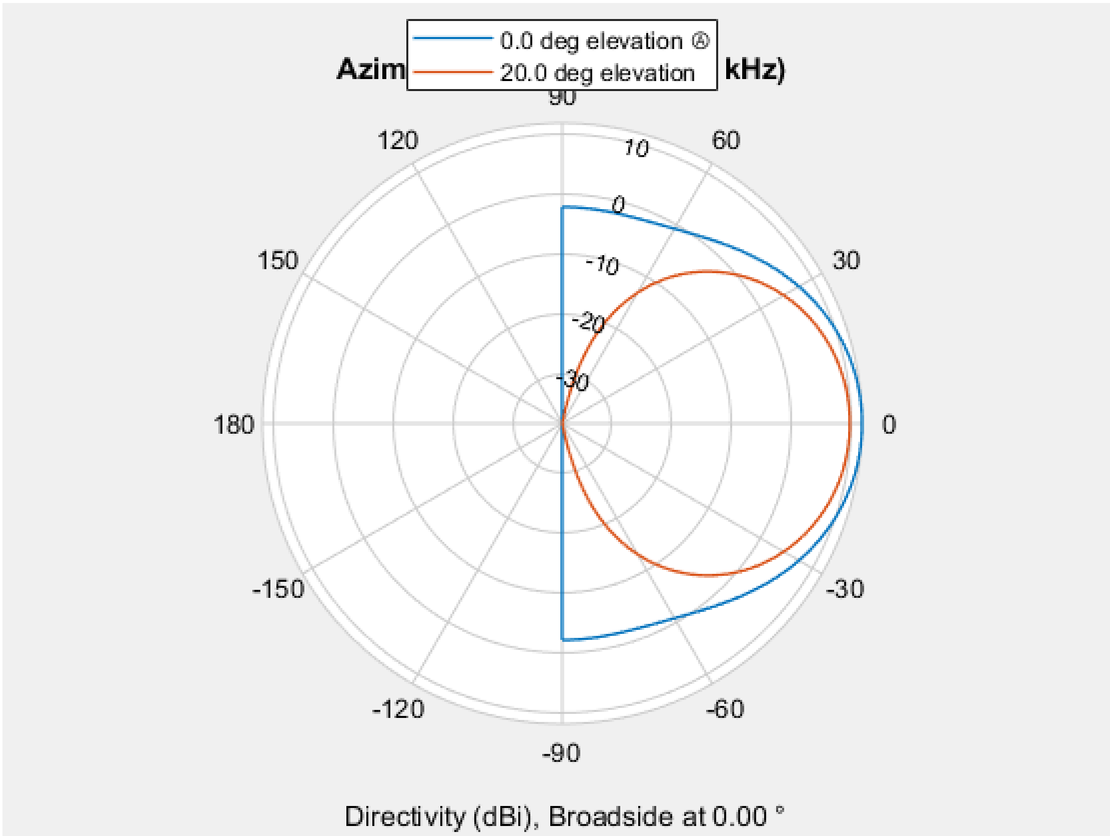
Plot azimuth pattern for magnitude

```
fc = 4000;
c = 1500.0;
patternAzimuth(sArray,fc,[0,20],...
    'PropagationSpeed',c,...
    'Type','efield')
```



Plot azimuth pattern for directivity

```
patternAzimuth(sArray,fc,[0,20],...
    'PropagationSpeed',c,...
    'Type','directivity')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also
`pattern` | `patternElevation`

patternElevation

System object: `phased.ConformalArray`
Package: `phased`

Plot conformal array directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sArray,FREQ)` plots the 2-D array directivity pattern versus elevation (in dBi) for the array `sArray` at zero degrees azimuth angle. When `AZ` is a vector, multiple overlaid plots are created. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternElevation(sArray,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sArray,FREQ,AZ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sArray` — Conformal array

System object

Conformal array, specified as a `phased.ConformalArray` System object.

Example: `sArray= phased.ConformalArray;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

`PAT` — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

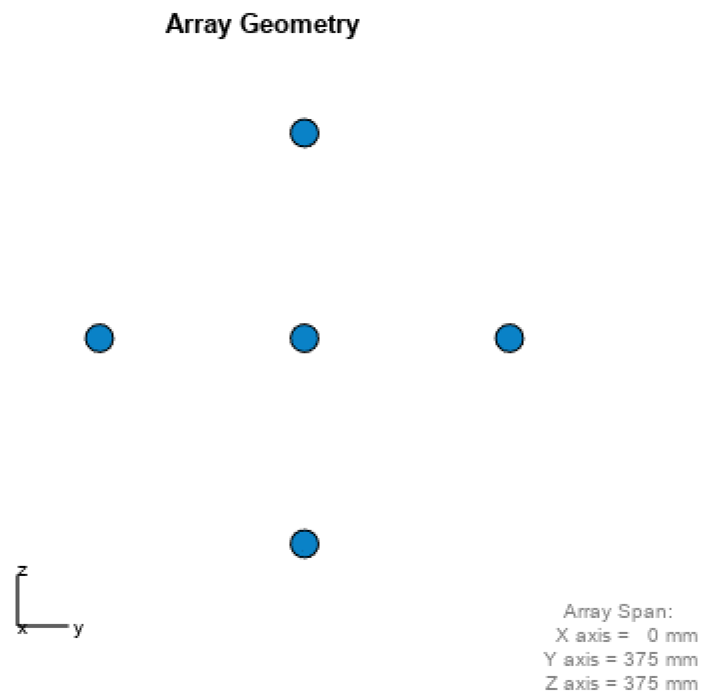
Examples

Plot Elevation Pattern of 5-Element Cross Sonar Array

Construct a 5-element acoustic cross array (UCA) using the ConformalArray System object™. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s. Plot the array patterns at two different azimuth angles.

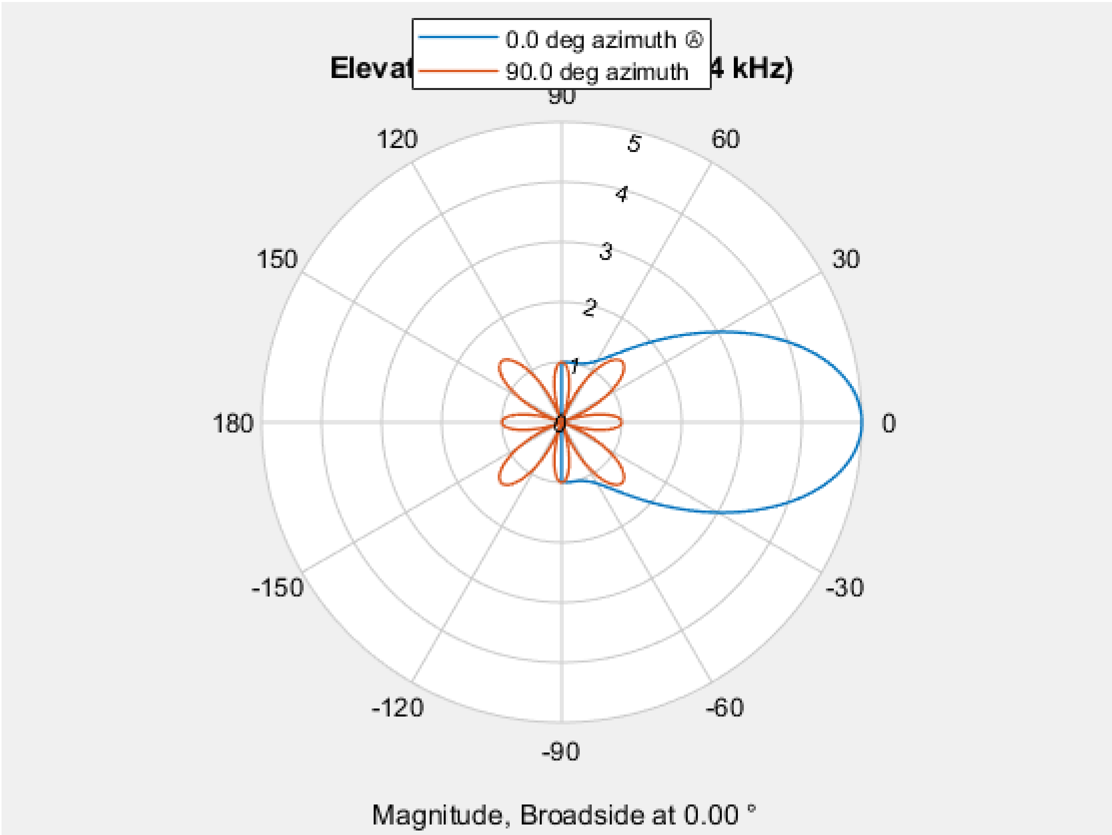
Construct and view array

```
N = 5;
fc = 4000;
c = 1500.0;
lam = c/fc;
x = zeros(1,N);
y = [-1,0,1,0,0]*lam/2;
z = [0,0,0,-1,1]*lam/2;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[0,10000],'BackBaffled',true);
sArray = phased.ConformalArray('Element',sMic,...
    'ElementPosition',[x;y;z],...
    'ElementNormal',[zeros(1,N);zeros(1,N)]);
viewArray(sArray)
```



Plot magnitude elevation pattern

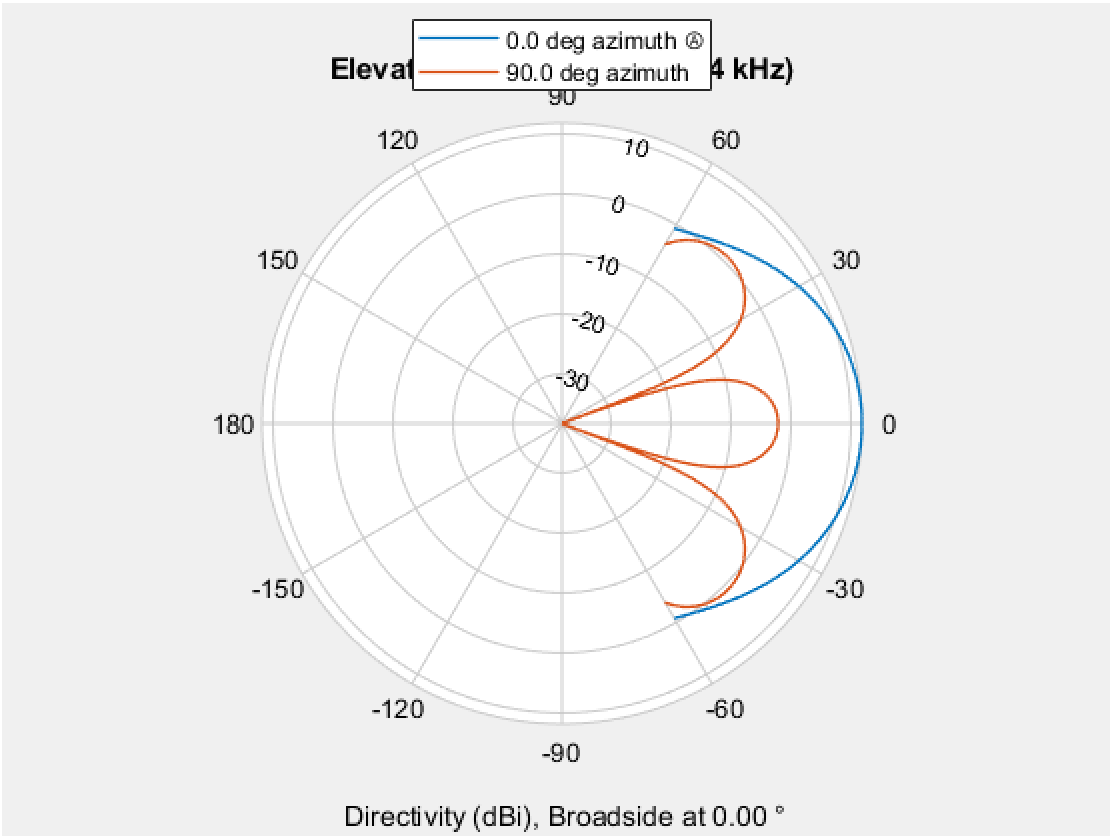
```
fc = 4000;
c = 1500.0;
patternElevation(sArray,fc,[0,90],...
    'PropagationSpeed',c,...
    'Type','efield')
```



Plot directivity elevation pattern

Plot the pattern for elevation angles between -60 and 6- degrees at 0.1 degree resolution.

```
patternElevation(sArray,fc,[0,90],...
    'PropagationSpeed',c,...
    'Type','directivity',...
    'Elevation',[-60:0.1:60])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth`

plotResponse

System object: `phased.ConformalArray`
Package: `phased`

Plot response pattern of array

Syntax

```
plotResponse(H,FREQ,V)
plotResponse(H,FREQ,V,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ,V)` plots the array response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`. The propagation speed is specified in `V`.

`plotResponse(H,FREQ,V,Name,Value)` plots the array response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Array object

FREQ

Operating frequency in Hertz specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CutAngle`

Cut angle as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

`Format`

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

`NormalizeResponse`

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

`OverlayFreq`

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

`Polarization`

Specify the polarization options for plotting the array response pattern. The allowable values are `|'None' | 'Combined' | 'H' | 'V' |` where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

`RespCut`

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.

- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

`Unit`

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

`Weights`

Weight values applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. The dimension *N* is the number of elements in the array. The interpretation of *M* depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

`AzimuthAngles`

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

`ElevationAngles`

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the

`Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

`UGrid`

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

`VGrid`

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

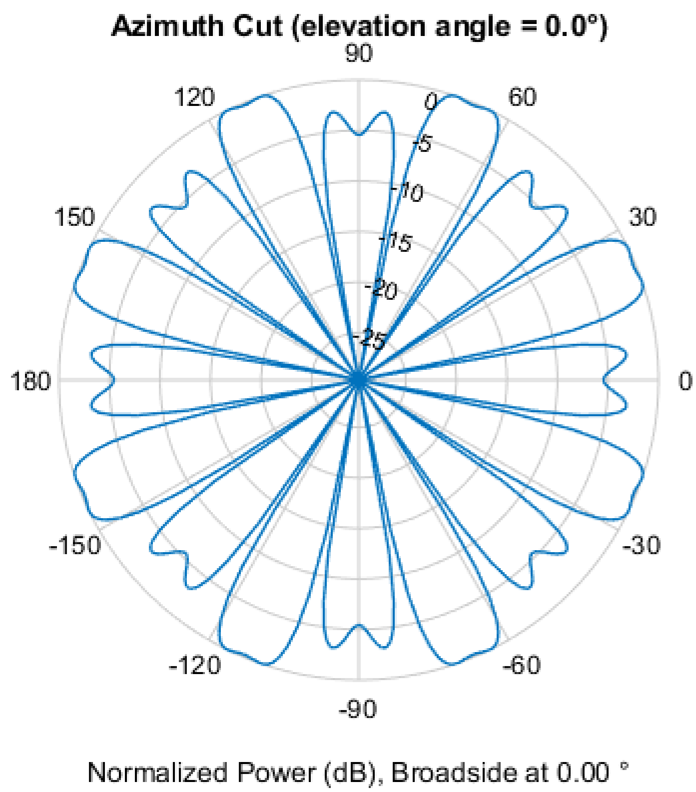
Default: `[-1:0.01:1]`

Examples

Plot Power Pattern of 8-Element Uniform Circular Array

Using the ConformalArray System object™, construct an 8-element uniform circular array (UCA) of isotropic antenna elements. Plot a normalized azimuth power pattern at 0 degrees elevation. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
N = 8;
azang = (0:N-1)*360/N-180;
sCA = phased.ConformalArray(...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
fc = 1e9;
c = physconst('LightSpeed');
pattern(sCA,fc,[-180:180],0,...
    'PropagationSpeed',c,'Type','powerdb',...
    'CoordinateSystem','polar')
```



Plot Pattern of 31-Element Uniform Circular Sonar Array

Construct a 31-element acoustic uniform circular sonar array (UCA) using the ConformalArray System object™. Assume the array is one meter in diameter. Using the `ElevationAngles` parameter, restrict the display to +/-40 degrees in 0.1 degree increments. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s.

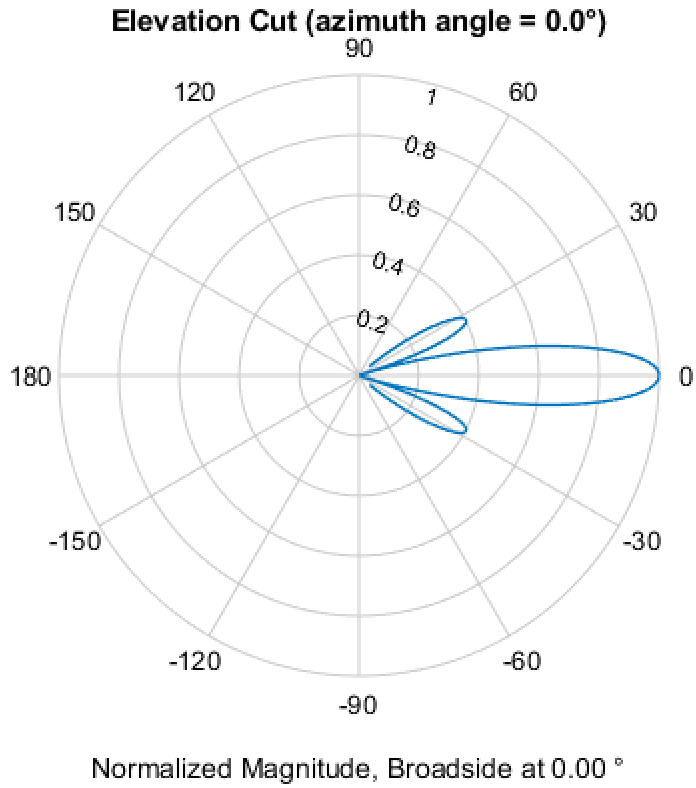
Construct the array

```
N = 31;
theta = (0:N-1)*360/N-180;
Radius = 0.5;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[0,10000],'BackBaffled',true);
sArray = phased.ConformalArray('Element',sMic,...
    'ElementPosition',Radius*[zeros(1,N);cosd(theta);sind(theta)],...
    'ElementNormal',[ones(1,N);zeros(1,N)]);
```

Plot the magnitude pattern

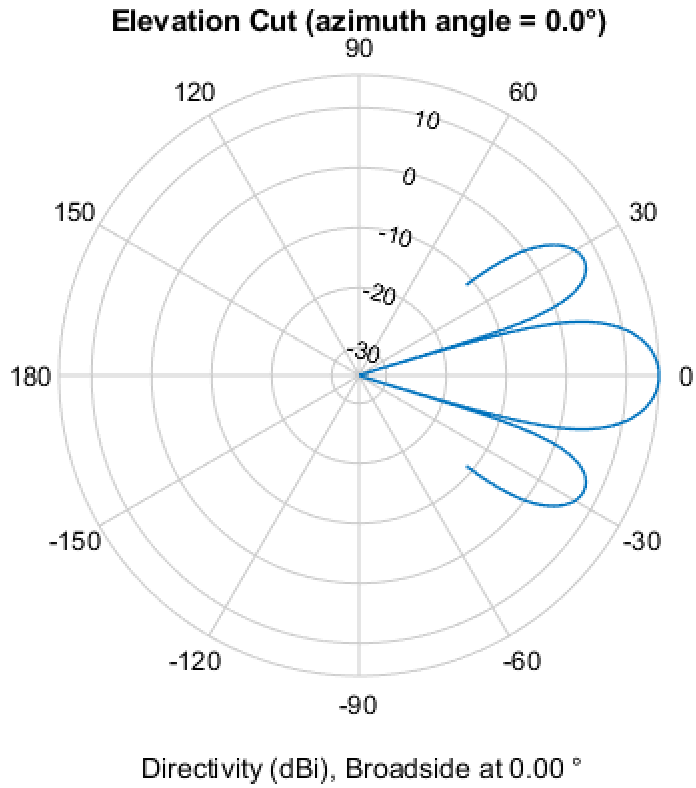
```
fc = 4000;
c = 1500.0;
pattern(sArray,fc,0,[-40:0.1:40],...
    'PropagationSpeed',c,...
```

```
    'CoordinateSystem','polar',...
    'Type','efield')
```



Plot the directivity pattern

```
pattern(sArray,fc,0,[-40:0.1:40],...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','directivity')
```



See Also

`uv2azel` | `azel2uv`

step

System object: `phased.ConformalArray`
Package: `phased`

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the response of the array elements, `RESP`, at operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`H`

Array object

`FREQ`

Operating frequencies of array in hertz. `FREQ` is a row vector of length *L*. Typical values are within the range specified by a property of `H.Element`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has zero response at frequencies outside that range.

`ANG`

Directions in degrees. `ANG` is either a 2-by-*M* matrix or a row vector of length *M*.

If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

If ANG is a row vector of length *M*, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be 0°.

Output Arguments

RESP

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, RESP, has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array. The dimension *M* is the number of angles specified in ANG. *L* is the number of frequencies specified in FREQ. For any element, the columns of RESP contain the responses of the array elements for the corresponding direction specified in ANG. Each of the *L* pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.
- If the array is capable of supporting polarization, the voltage response, RESP, is a MATLAB struct containing two fields, RESP.H and RESP.V. The field, RESP.H, represents the array's horizontal polarization response, while RESP.V represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array, and *M* is the number of angles specified in ANG. *L* is the number of frequencies specified in FREQ. Each column of RESP contains the responses of the array elements for the corresponding direction specified in ANG. Each of the *L* pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.

Examples

Response of 8-Element Uniform Circular Array

Using the ConformalArray System object™, construct an 8-element uniform circular array (UCA) of isotropic antenna elements. The radius of the array is one meter. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
N = 8;
azang = (0:N-1)*360/N-180;
sCA = phased.ConformalArray(...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
```

Get the element response at 35 degrees azimuth and 5 degrees elevation.

```
fc = 1e9;
ang = [30;5];
resp = step(sCA,fc,ang)
```

```
resp = 8×1

     1
     1
     1
     1
     1
     1
```

```
1
1
```

See Also

uv2azel | phitheta2azel

viewArray

System object: `phased.ConformalArray`
Package: `phased`

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

`viewArray(H)` plots the geometry of the array specified in `H`.

`viewArray(H,Name,Value)` plots the geometry of the array, with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = viewArray( ___ )` returns the handle of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

`H`

Array object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Parent`

Handle to the axes along which the array geometry is displayed.

`ShowNormals`

Set this value to `true` to show the normal directions of all elements of the array. Set this value to `false` to plot the elements without showing normal directions.

Default: `false`

`ShowLocalCoordinates`

Logical flag specifying whether to show the local coordinate axes.

Default: `true`

`ShowAnnotation`

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Default: `true`

`Orientation`

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Default: `[0;0;0]`

`ShowTaper`

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

`ShowIndex`

Vector specifying the element indices to show in the figure. Each number in the vector must be an integer between 1 and the number of elements. You can also specify the value `'All'` to show the indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

`Title`

Character vector specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

`hPlot`

Handle of array elements in figure window.

Examples

View Uniform Circular Array

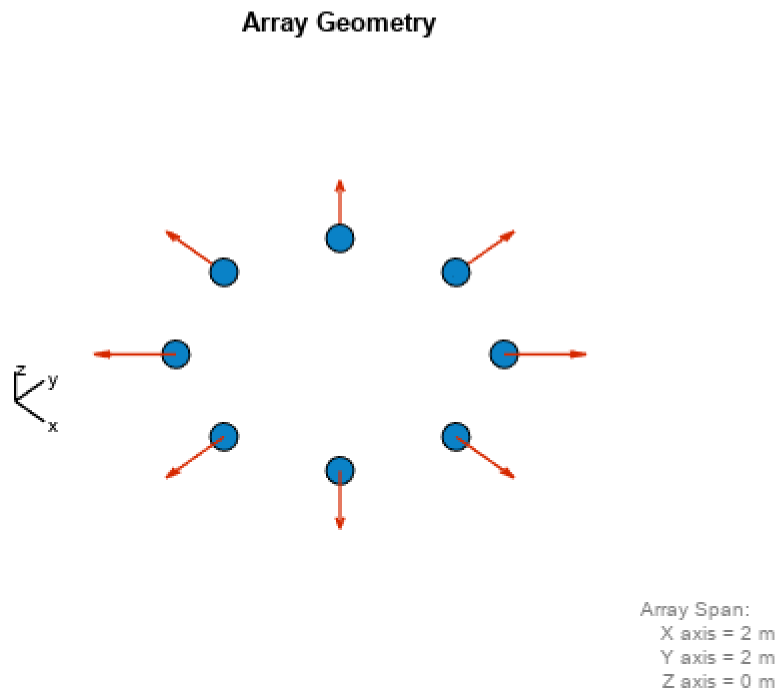
Display the element positions and normal directions of all elements of an 8-element uniform circular array.

Create the uniform circular array

```
N = 8;
azang = (0:N-1)*360/N - 180;
ha = phased.ConformalArray(...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
```

Display the positions and normal directions of the elements

```
viewArray(ha,'ShowNormals',true);
```



See Also

phased.ArrayResponse

Topics

Phased Array Gallery

phased.CardioidAntennaElement

Package: `phased`

Cardioid antenna element

Description

The `CardioidAntennaElement` System object models an antenna with a "Cardioid Response" on page 1-221. Cardioid antennas are often used in direction finding. The cardioid response can be implemented by placing two isotropic radiators in an array, a quarter-wavelength apart and 90° out of phase. The default rotation of the cardioid pattern has a null at 180° azimuth and 0° elevation. The 0° azimuth and 0° elevation is considered to be the main response axis of the antenna. When placed in a linear or a rectangular array, the main response axis is aligned with the array normal.

To compute the response of the antenna element for specified directions:

1. Create the `phased.CardioidAntennaElement` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
antenna = phased.CardioidAntennaElement
antenna = phased.CardioidAntennaElement(Name=Value)
```

Description

`antenna = phased.CardioidAntennaElement` creates a cardioid antenna System object, `antenna`. This object models an antenna element whose response is a cardioid with a null at 180° azimuth and 0° elevation.

`antenna = phased.CardioidAntennaElement(Name=Value)` creates a cardioid antenna object, `antenna`, with each specified property set to the specified value. You can specify multiple name-value arguments in any order. For example, `FrequencyRange=[1e6 1e9]` specifies that the antenna operates in a frequency range from 1 MHz to 1 GHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

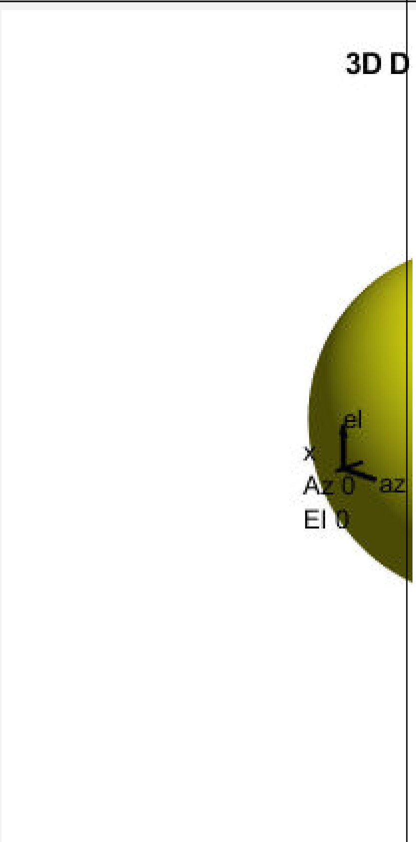
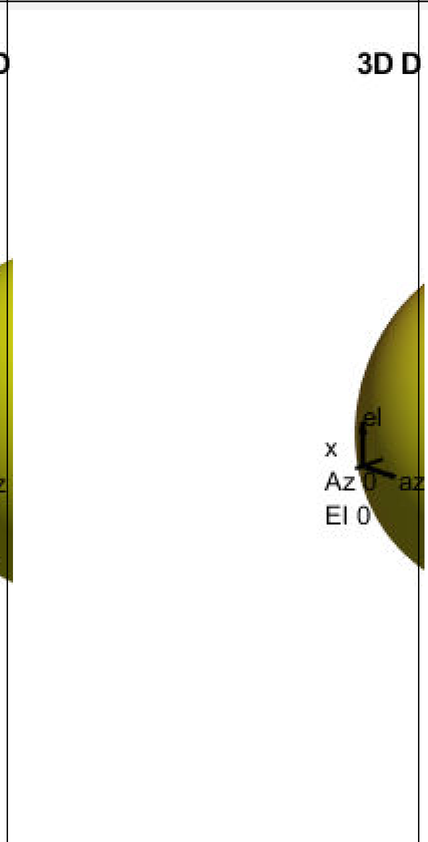
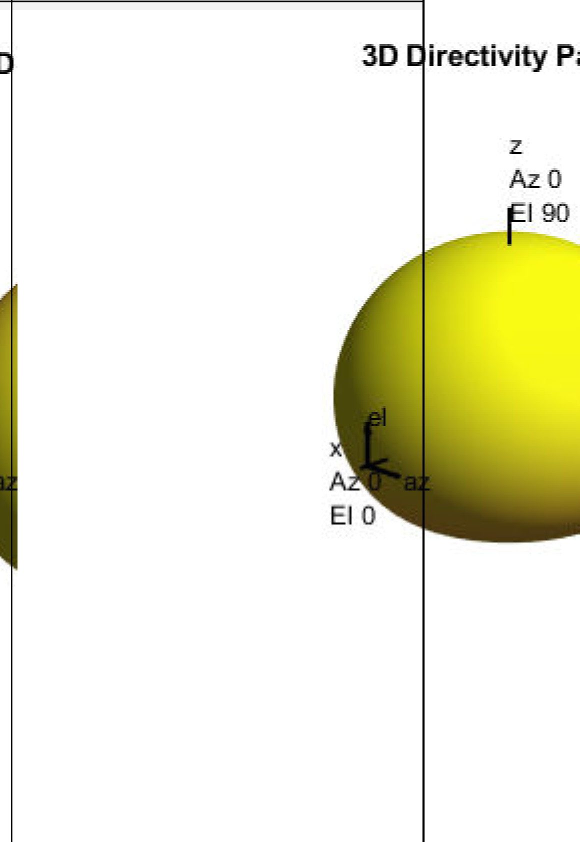
FrequencyRange — Operating frequency range
`[0 1e20]` (default) | nonnegative, real-valued 1-by-2 row vector

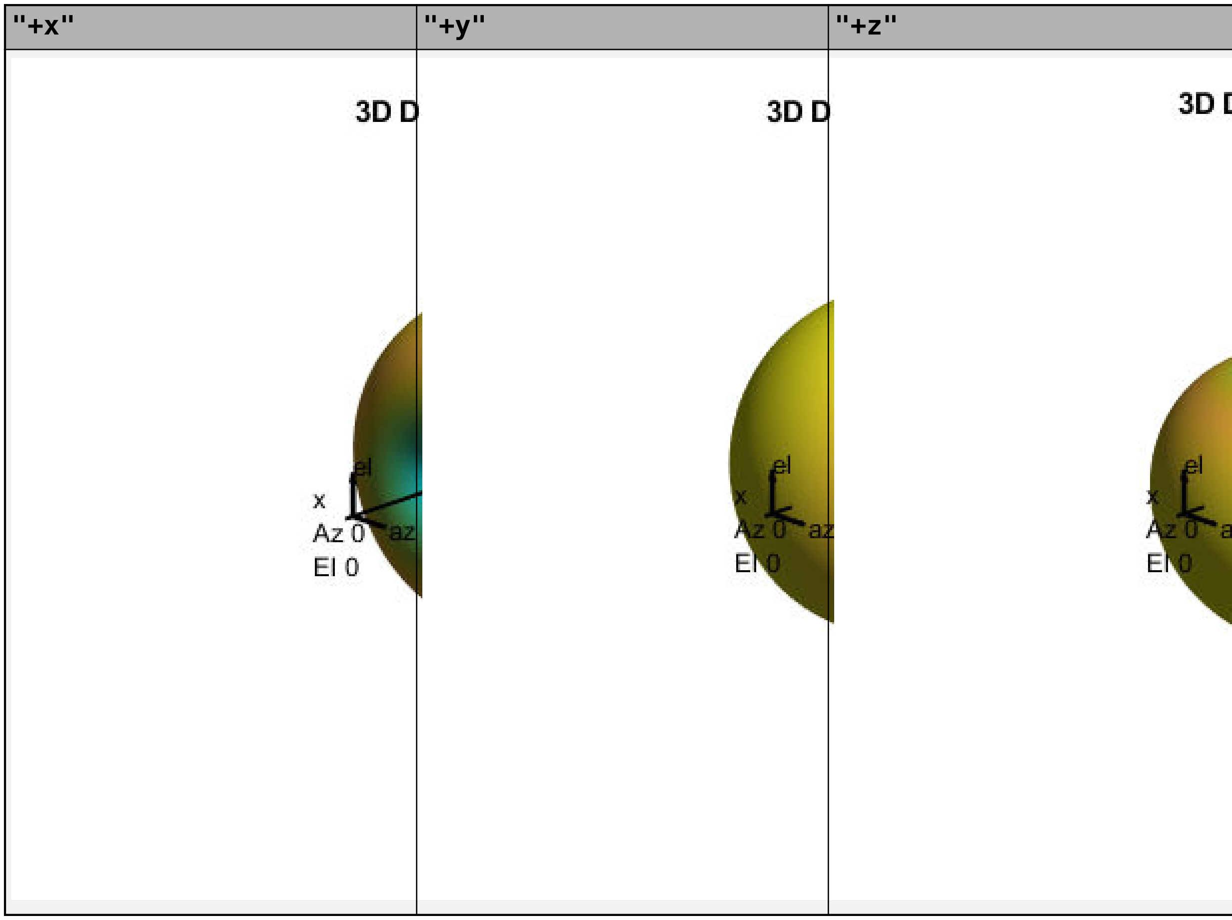
Operating frequency range of the antenna, specified as a nonnegative, real-valued, 1-by-2 row vector in the form `[LowerBound HigherBound]`. The antenna element has no response outside the specified frequency range. Units are in Hz.

Data Types: `double`

NullAxisDirection — Null axis direction
`"-x"` (default) | `"+x"` | `"-y"` | `"+y"` | `"-z"` | `"+z"`

Null axis direction, specified as one of these:

"-x"	"-y"	"-z"
		



For more information, see “Cardioid Response” on page 1-221.

Data Types: `double`

Usage

Syntax

`RESP = antenna(FREQ,ANG)`

Description

`RESP = antenna(FREQ,ANG)` returns the antenna voltage response `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

FREQ — Operating frequency of antenna element
nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

`FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the response is returned as `–Inf`. Element objects use the `FrequencyRange` property, except for `phased.CustomAntennaElement`, which uses the `FrequencyVector` property.

Example: `[1e8 2e6]`

Data Types: `double`

ANG — Azimuth and elevation angles of response directions
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If `ANG` is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of "Azimuth and Elevation Angles".

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

RESP — Voltage response of antenna
M-by-*L* matrix

Voltage response of the antenna element, returned as a matrix of size *M*-by-*L*. In this matrix, *M* represents the number of angles specified in `ANG` and *L* represents the number of frequencies specified in `FREQ`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

patternElevation	Plot antenna or transducer element directivity and pattern versus elevation
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
pattern	Plot antenna or transducer element directivity and patterns
beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability

Common to All System Objects

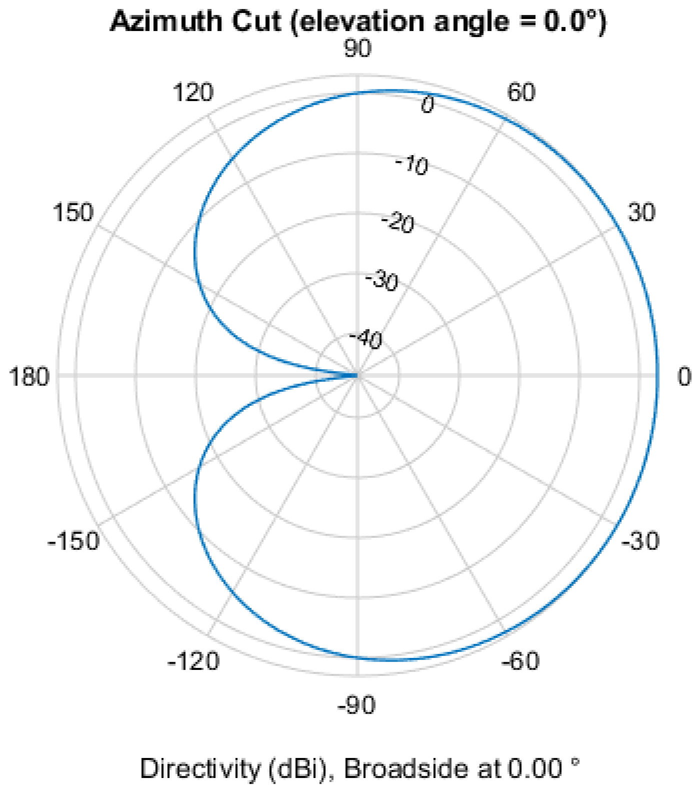
step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Cardioid Antenna Response

Create a cardioid antenna and plot its azimuth response. The antenna can work between 800 MHz and 1.2 GHz and has an operating frequency of 1 GHz.

```
element = phased.CardioidAntennaElement( ...
    FrequencyRange=[800e6 1.2e9]);
fc = 1e9;

pattern(element,fc,-180:180,0,CoordinateSystem="polar")
```



Find the response of the antenna at the boresight.

```
ang = [0 0]';
resp = element(fc,ang)
```

```
resp = 1
```

More About

Cardioid Response

The `phased.CardioidAntennaElement` object returns the field response (also called field pattern) of the cardioid antenna element.

If *az* is the azimuth angle in degrees and *el* is the elevation angle in degrees, the field response has a different expression depending on the value specified for `NullAxisDirection`.

"-x"	"-y"	"-z"
$f(az,el) = \dfrac{\sin\left(\frac{\pi}{2}\sin(az-90°)\cos(el)+\frac{\pi}{2}\right)}{2\sin\left(\frac{1}{2}\left(\frac{\pi}{2}\sin(az-90°)\cos(el)+\frac{\pi}{2}\right)\right)}$	$f(az,el) = \dfrac{\sin\left(\frac{\pi}{2}\sin(az-180°)\cos(el)+\frac{\pi}{2}\right)}{2\sin\left(\frac{1}{2}\left(\frac{\pi}{2}\sin(az-180°)\cos(el)+\frac{\pi}{2}\right)\right)}$	$f(az,el) = \dfrac{\sin\left(\frac{\pi}{2}\sin(-el)+\frac{\pi}{2}\right)}{2\sin\left(\frac{1}{2}\left(\frac{\pi}{2}\sin(-el)+\frac{\pi}{2}\right)\right)}$

"+x"	"+y"	"+z"
$f(az,el) = \dfrac{\sin\left(\frac{\pi}{2}\sin(az+90°)\cos(el)+\frac{\pi}{2}\right)}{2\sin\left(\frac{1}{2}\left(\frac{\pi}{2}\sin(az+90°)\cos(el)+\frac{\pi}{2}\right)\right)}$	$f(az,el) = \dfrac{\sin\left(\frac{\pi}{2}\sin(az)\cos(el)+\frac{\pi}{2}\right)}{2\sin\left(\frac{1}{2}\left(\frac{\pi}{2}\sin(az)\cos(el)+\frac{\pi}{2}\right)\right)}$	$f(az,el) = \dfrac{\sin\left(\frac{\pi}{2}\sin(el)+\frac{\pi}{2}\right)}{2\sin\left(\frac{1}{2}\left(\frac{\pi}{2}\sin(el)+\frac{\pi}{2}\right)\right)}$

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `pattern`, `patternAzimuth`, and `patternElevation` object functions are not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.ConformalArray` | `phased.CustomAntennaElement` | `phased.CosineAntennaElement` | `phased.GaussianAntennaElement` | `phased.IsotropicAntennaElement` | `phased.SincAntennaElement` | `phased.ULA` | `phased.URA`

phased.GaussianAntennaElement

Package: `phased`

Gaussian antenna element

Description

The `GaussianAntennaElement` System object models an antenna with a “Gaussian Response” on page 1-226. Despite being an idealized antenna pattern, the Gaussian is often used to approximate other antennas in simulations because its response closely follows the pattern of many antennas out to about the –10 dB level. The Gaussian beam has no sidelobes. The 0° azimuth and 0° elevation is considered to be the main response axis of the antenna. When placed in a linear or a rectangular array, the main response axis is aligned with the array normal.

To compute the response of the antenna element for specified directions:

1. Create the `phased.GaussianAntennaElement` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
antenna = phased.GaussianAntennaElement
antenna = phased.GaussianAntennaElement(Name=Value)
```

Description

`antenna = phased.GaussianAntennaElement` creates a Gaussian antenna System object, `antenna`. This object models a Gaussian beam, which closely follows the pattern of many antennas out to about the –10 dB level. The Gaussian beam has no sidelobes.

`antenna = phased.GaussianAntennaElement(Name=Value)` creates a Gaussian antenna object, `antenna`, with each specified property set to the specified value. You can specify multiple name-value arguments in any order. For example, `FrequencyRange=[1e6 1e9]` specifies that the antenna operates in a frequency range from 1 MHz to 1 GHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`FrequencyRange` — Operating frequency range
`[0 1e20]` (default) | nonnegative, real-valued 1-by-2 row vector

Operating frequency range of the antenna, specified as a nonnegative, real-valued, 1-by-2 row vector in the form `[LowerBound HigherBound]`. The antenna element has no response outside the specified frequency range. Units are in Hz.

Data Types: `double`

`Beamwidth` — Beamwidth of antenna pattern
`[10 10]` (default) | scalar | 1-by-2 real-valued vector

Beamwidth of the antenna pattern, specified as either a scalar or a 1-by-2 real-valued vector. When the specified value is a 1-by-2 vector, it has the form of `[AzimuthBeamwidth ElevationBeamwidth]`. If the specified value is a scalar, the azimuth and elevation beamwidths are equal. Units are in degrees.

Example: `15`

Data Types: `double`

Usage

Syntax

`RESP = antenna(FREQ,ANG)`

Description

`RESP = antenna(FREQ,ANG)` returns the antenna voltage response `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`.

Input Arguments

`FREQ` — Operating frequency of antenna element
nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

`FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the response is returned as `–Inf`. Element objects use the `FrequencyRange` property, except for `phased.CustomAntennaElement`, which uses the `FrequencyVector` property.

Example: `[1e8 2e6]`

Data Types: `double`

`ANG` — Azimuth and elevation angles of response directions
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If `ANG` is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of “Azimuth and Elevation Angles”.

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

RESP — Voltage response of antenna
M-by-*L* matrix

Voltage response of the antenna element, returned as a matrix of size *M*-by-*L*. In this matrix, *M* represents the number of angles specified in `ANG` and *L* represents the number of frequencies specified in `FREQ`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

directivity	Directivity of antenna or transducer element
beamwidth	Compute and display beamwidth of sensor element pattern
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation
isPolarizationCapable	Antenna element polarization capability

Common to All System Objects

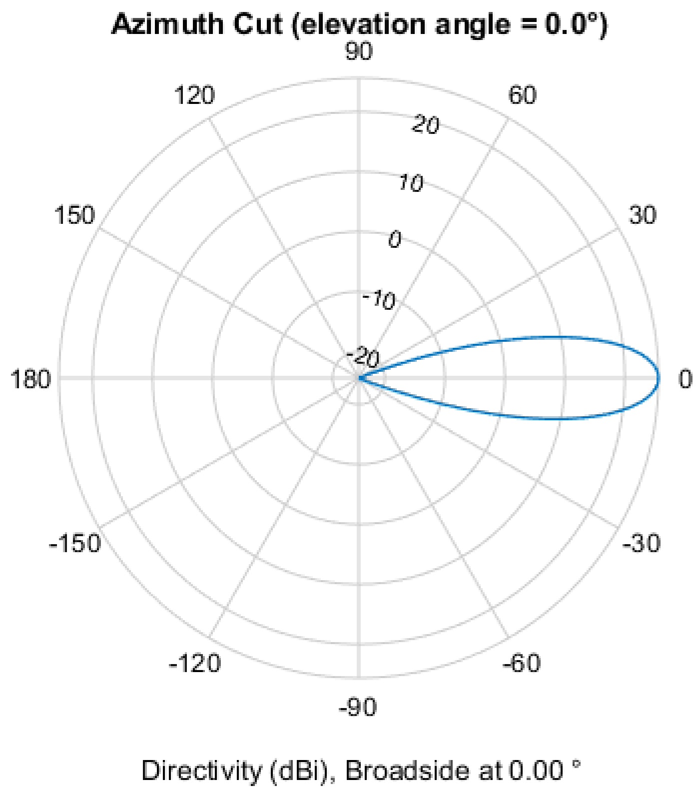
step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Gaussian Antenna Response

Create a Gaussian antenna and plot its azimuth response. The antenna can work between 800 MHz and 1.2 GHz and has an operating frequency of 1 GHz.

```
element = phased.GaussianAntennaElement( ...
    FrequencyRange=[800e6 1.2e9]);
fc = 1e9;

pattern(element,fc,-180:180,0,CoordinateSystem="polar")
```



Find the response of the antenna at the boresight.

```
ang = [0 0]';
resp = element(fc,ang)
```

```
resp = 1
```

More About

Gaussian Response

The `phased.GaussianAntennaElement` object returns the field response (also called field pattern) of the Gaussian antenna element.

If *az* is the azimuth angle in degrees and *el* is the elevation angle in degrees, the field response is

$$f(az, el) = \exp\left(-2\log 2\left(\frac{az}{\mathrm{HPBW}_{az}}\right)^2\right)\exp\left(-2\log 2\left(\frac{el}{\mathrm{HPBW}_{el}}\right)^2\right),$$

where the azimuth half-power beamwidth $HPBW_{az}$ is specified as the first element of `Beamwidth` and the elevation half-power beamwidth $HPBW_{el}$ is specified as the second element of `Beamwidth`.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `pattern`, `patternAzimuth`, and `patternElevation` object functions are not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.CardioidAntennaElement` | `phased.ConformalArray` | `phased.CustomAntennaElement` | `phased.CosineAntennaElement` | `phased.IsotropicAntennaElement` | `phased.SincAntennaElement` | `phased.ULA` | `phased.URA`

Topics

"Gaussian Antenna as Approximation for Spiral Antenna"

phased.SincAntennaElement

Package: phased

Sinc antenna element

Description

The SincAntennaElement System object models an antenna with a “Sinc Response” on page 1-231. A sinc antenna is representative of a uniformly illuminated rectangular antenna. Sinc antenna patterns are often used as an approximation for sector or array antennas. The sidelobes are –13.6 dB relative to the main beam gain. The 0° azimuth and 0° elevation is considered to be the main response axis of the antenna. When placed in a linear or a rectangular array, the main response axis is aligned with the array normal.

To compute the response of the antenna element for specified directions:

1. Create the phased.SincAntennaElement object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
antenna = phased.SincAntennaElement
antenna = phased.SincAntennaElement(Name=Value)
```

Description

antenna = phased.SincAntennaElement creates a sinc antenna System object, antenna. This object models an antenna element whose response is a sinc antenna, which is representative of a uniformly illuminated rectangular antenna. The sidelobes are –13.6 dB relative to the main beam gain.

antenna = phased.SincAntennaElement(Name=Value) creates a sinc antenna object, antenna, with each specified property set to the specified value. You can specify multiple name-value arguments in any order. For example, FrequencyRange=[1e6 1e9] specifies that the antenna operates in a frequency range from 1 MHz to 1 GHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

FrequencyRange — Operating frequency range
[0 1e20] (default) | nonnegative, real-valued 1-by-2 row vector

Operating frequency range of the antenna, specified as a nonnegative, real-valued, 1-by-2 row vector in the form [LowerBound HigherBound]. The antenna element has no response outside the specified frequency range. Units are in Hz.

Data Types: double

Beamwidth — Beamwidth of antenna pattern
[10 10] (default) | scalar | 1-by-2 real-valued vector

Beamwidth of the antenna pattern, specified as either a scalar or a 1-by-2 real-valued vector. When the specified value is a 1-by-2 vector, it has the form of [AzimuthBeamwidth ElevationBeamwidth]. If the specified value is a scalar, the azimuth and elevation beamwidths are equal. Units are in degrees.

Example: 15

Data Types: double

Usage

Syntax

RESP = antenna(FREQ,ANG)

Description

RESP = antenna(FREQ,ANG) returns the antenna voltage response RESP at operating frequencies specified in FREQ and directions specified in ANG.

Input Arguments

FREQ — Operating frequency of antenna element
nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element produces no response and the response is returned as –Inf. Element objects use the FrequencyRange property, except for phased.CustomAntennaElement, which uses the FrequencyVector property.

Example: [1e8 2e6]

Data Types: double

ANG — Azimuth and elevation angles of response directions
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If ANG is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If ANG is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of "Azimuth and Elevation Angles".

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

RESP — Voltage response of antenna
M-by-*L* matrix

Voltage response of the antenna element, returned as a matrix of size *M*-by-*L*. In this matrix, *M* represents the number of angles specified in ANG and *L* represents the number of frequencies specified in FREQ.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

directivity	Directivity of antenna or transducer element
beamwidth	Compute and display beamwidth of sensor element pattern
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation
isPolarizationCapable	Antenna element polarization capability

Common to All System Objects

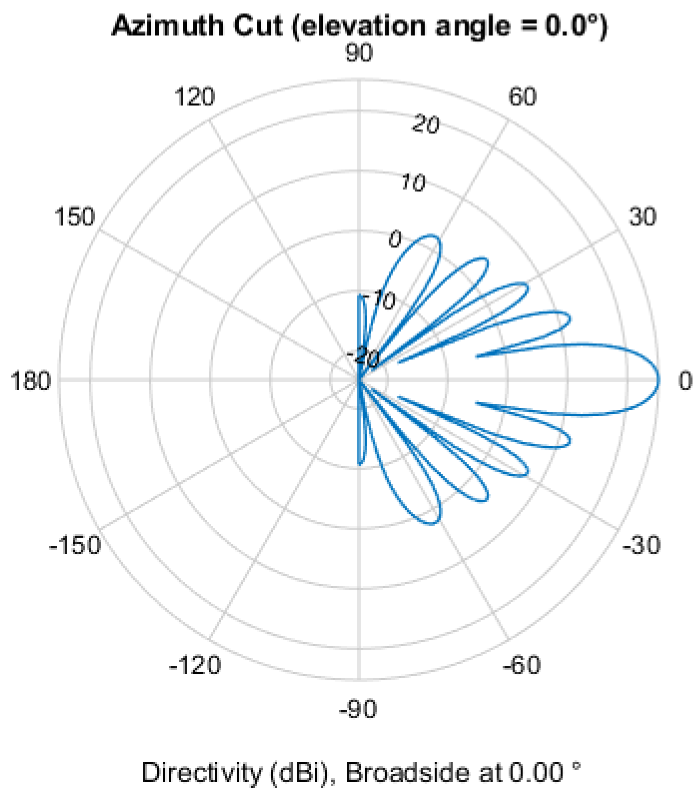
step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Sinc Antenna Response

Create a sinc antenna and plot its azimuth response. The antenna can work between 800 MHz and 1.2 GHz and has an operating frequency of 1 GHz.

```
element = phased.SincAntennaElement( ...
    FrequencyRange=[800e6 1.2e9]);
fc = 1e9;

pattern(element,fc,-180:180,0,CoordinateSystem="polar")
```



Find the response of the antenna at the boresight.

```
ang = [0 0]';
resp = element(fc,ang)
```

```
resp = 1
```

More About

Sinc Response

The `phased.SincAntennaElement` object returns the field response (also called field pattern) of the sinc antenna element.

If *az* is the azimuth angle in degrees and *el* is the elevation angle in degrees, the field response is

$$f(az, el) = \frac{\sin(u_{az})}{u_{az}} \frac{\sin(u_{el})}{u_{el}},$$

where:

- The azimuth factor u_{az} depends on *az* and the azimuth half-power beamwidth HPBW_{az}:

  $$u_{az} = \frac{x_0 \sin(az)}{\sin\left(\frac{1}{2} \times \mathrm{HPBW}_{az}\right)}.$$

  Specify HPBW_{az} as the first element of `Beamwidth`.
- The elevation factor u_{el} depends on *el* and the elevation half-power beamwidth HPBW_{el}:

  $$u_{el} = \frac{x_0 \sin(el)}{\sin\left(\frac{1}{2} \times \mathrm{HPBW}_{el}\right)}.$$

  Specify HPBW_{el} as the second element of `Beamwidth`.
- The normalization factor x_0 ensures the power is ½ at the half-power beamwidth angle. The factor is a solution of

  $$\sin x = \frac{x}{\sqrt{2}}.$$

Version History

Introduced in R2021b

References

[1] Blake, Lamont V. *Machine Plotting of Radar Vertical-Plane Coverage Diagrams*. Naval Research Laboratory Report 7098, 1970.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `pattern`, `patternAzimuth`, and `patternElevation` object functions are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.CardioidAntennaElement` | `phased.ConformalArray` | `phased.CustomAntennaElement` | `phased.CosineAntennaElement` | `phased.GaussianAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.URA`

Topics

“Sinc Antenna as Approximation for Array Response Pattern”

phased.CosineAntennaElement

Package: phased

Cosine antenna element

Description

The CosineAntennaElement object models an antenna with a cosine response on page 1-245 in both azimuth and elevation. The main response axis (MRA) points to 0° azimuth and 0° elevation in the antenna coordinate system. When placed in a linear array, the MRA is normal to the array axis (see, for example, phased.ULA). When placed in a planar array, the MRA points along the array normal (see, for example, phased.URA).

To compute the response of the antenna element for specified directions:

1 Create the phased.CosineAntennaElement object and set its properties.
2 Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

This antenna element does not support polarization.

Creation

Syntax

```
antenna = phased.CosineAntennaElement
antenna = phased.CosineAntennaElement(Name,Value)
```

Description

antenna = phased.CosineAntennaElement creates a cosine antenna System object, antenna. This object models an antenna element whose response is a cosine function raised to nonnegative powers in the azimuth and elevation directions.

antenna = phased.CosineAntennaElement(Name,Value) creates a cosine antenna object, antenna, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

FrequencyRange — Operating frequency range

[0 1e20] (default) | nonnegative, real-valued 1-by-2 row vector

Operating frequency range of the antenna, specified as a nonnegative, real-valued, 1-by-2 row vector in the form [LowerBound HigherBound]. The antenna element has no response outside the specified frequency range. Units are in Hz.

Data Types: double

CosinePower — Exponent of cosine pattern

[1.5 1.5] (default) | non-negative scalar | non-negative, real-valued, 1-by-2 vector

Exponents of the cosine pattern, specified as a non-negative scalar or a non-negative, real-valued, 1-by-2 vector. Exponent values must be real numbers greater than or equal to zero. When you set CosinePower to a scalar, both the azimuth direction cosine pattern and the elevation direction cosine pattern are raised to the same power. When you set CosinePower to a 1-by-2 vector, the first element is the exponent for the azimuth direction cosine pattern. The second element is the exponent for the elevation direction cosine pattern.

Example: [1.5 1.3]

Data Types: double

Usage

Syntax

```
RESP = antenna(FREQ,ANG)
```

Description

RESP = antenna(FREQ,ANG) returns the antenna voltage response RESP at operating frequencies specified in FREQ and directions specified in ANG.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Input Arguments

FREQ — Operating frequency of antenna element

nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element produces no response and the response is returned as –Inf. Element objects use the FrequencyRange property, except for phased.CustomAntennaElement, which uses the FrequencyVector property.

Example: `[1e8 2e6]`

Data Types: `double`

`ANG` — Azimuth and elevation angles of response directions
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If `ANG` is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of “Azimuth and Elevation Angles”.

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

`RESP` — Voltage response of antenna
complex-valued *M*-by-*L* matrix

Voltage response of antenna element, returned as a complex-valued *M*-by-*L* matrix. In this matrix, *M* represents the number of angles specified in `ANG` and *L* represents the number of frequencies specified in `FREQ`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Cosine Antenna Response

Construct a cosine antenna element and find its response in one direction. The cosine response is raised to a power of 1.5 in both azimuth and elevation. The antenna frequency range lies in the X band (from 8 to 12 GHz) at 10 GHz. Obtain the antenna's response for an incident angle of 30° azimuth and 5° elevation.

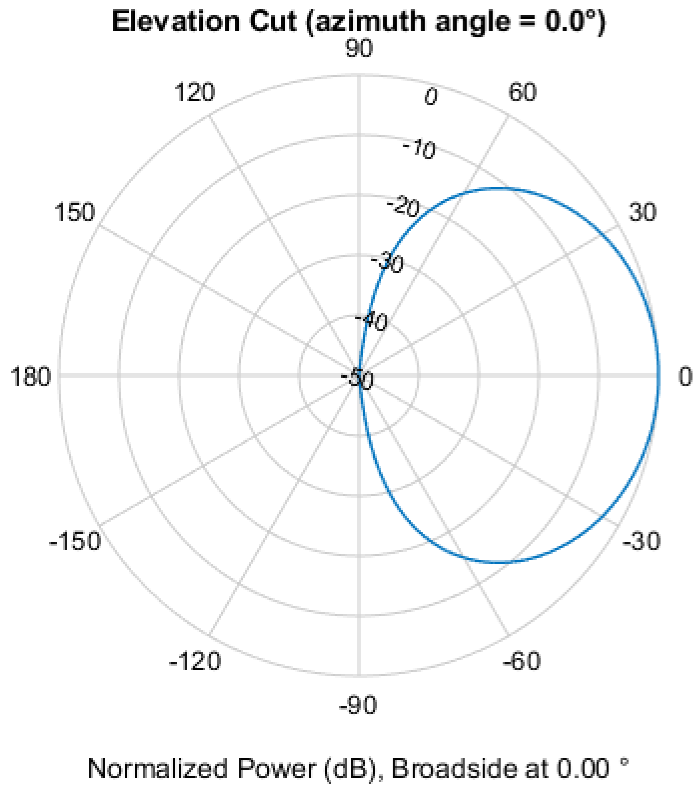
```
antenna = phased.CosineAntennaElement('FrequencyRange',[8e9 12e9], ...
    'CosinePower',1.5);
fc = 10.0e9;
ang = [30;5];
resp = antenna(fc,ang)
```

```
resp = 0.8013
```

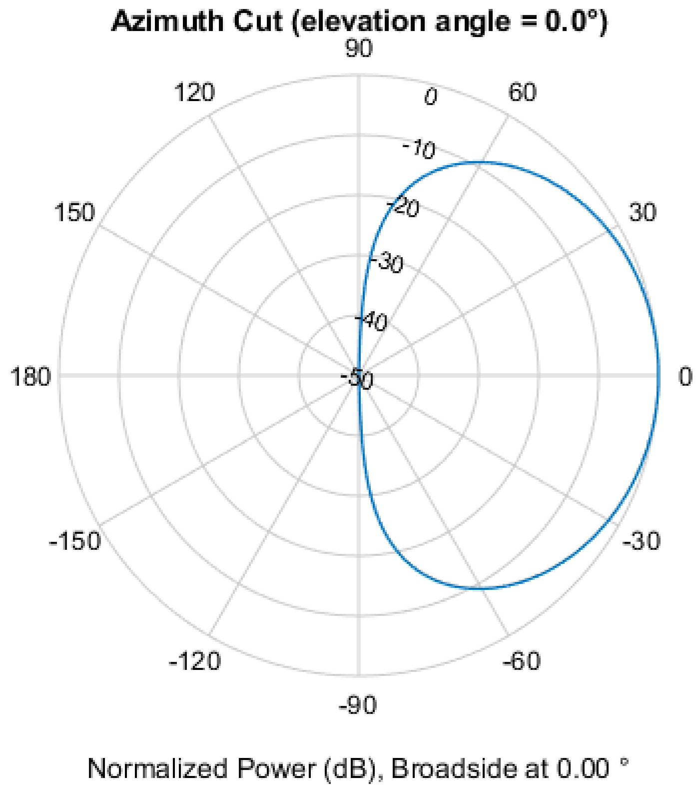
Plot Power Response of Cosine Antenna

Construct a cosine pattern antenna and calculate its response at boresight (0 degrees azimuth and 0 degrees elevation). Then, plot the antenna pattern. Assume the antenna works between 800 MHz and 1.2 GHz and its operating frequency is 1 GHz. Set the azimuth exponent to 1.5 and elevation exponent to 2.5.

```
antenna = phased.CosineAntennaElement('FrequencyRange',[800e6 1.2e9],...
    'CosinePower',[1.5 2.5]);
fc = 1e9;
resp = antenna(fc,[0;0]);
pattern(antenna,fc,0,-90:90,'Type','powerdb','CoordinateSystem','polar')
```



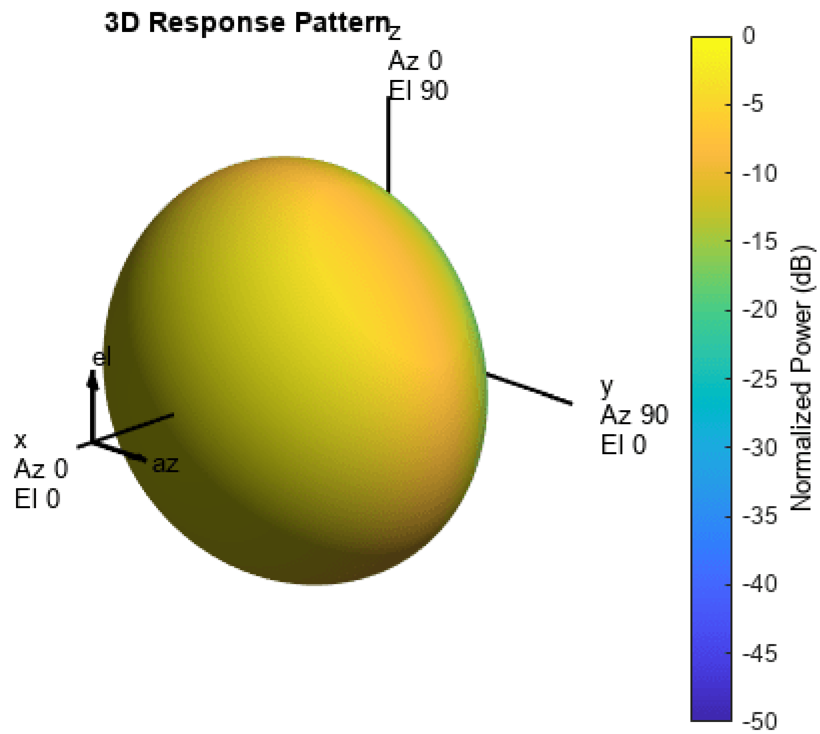

```
pattern(antenna,fc,-180:180,0,'Type','powerdb','CoordinateSystem','polar')
```



Plot 3-D Polar Pattern of Cosine Antenna

Construct a cosine antenna element using default parameters. Assume the antenna operating frequency is 1 GHz. Then, plot the antenna response in 3-D polar format.

```
antenna = phased.CosineAntennaElement;
fc = 1e9;
pattern(antenna,fc,[-180:180],[-90:90],'Type','powerdb', ...
    'CoordinateSystem','polar')
```



Directivity of Cosine Antenna

Compute the directivity of a cosine antenna element at seven azimuth directions centered around boresight (zero degrees azimuth and zero degrees elevation). All elevation angles are set to zero degrees.

Create a cosine antenna element system object with the `CosinePower` exponents set to 1.8.

```
antenna = phased.CosineAntennaElement('CosinePower',[1.8,1.8]);
```

Set the directivity angles so that the elevation angles are zero. Set the frequency to 1 GHz.

```
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
freq = 1e9;
```

Compute the directivity.

```
d = directivity(antenna,freq,ang)
```

```
d = 7×1

    7.3890
    8.6654
    9.3985
    9.6379
```

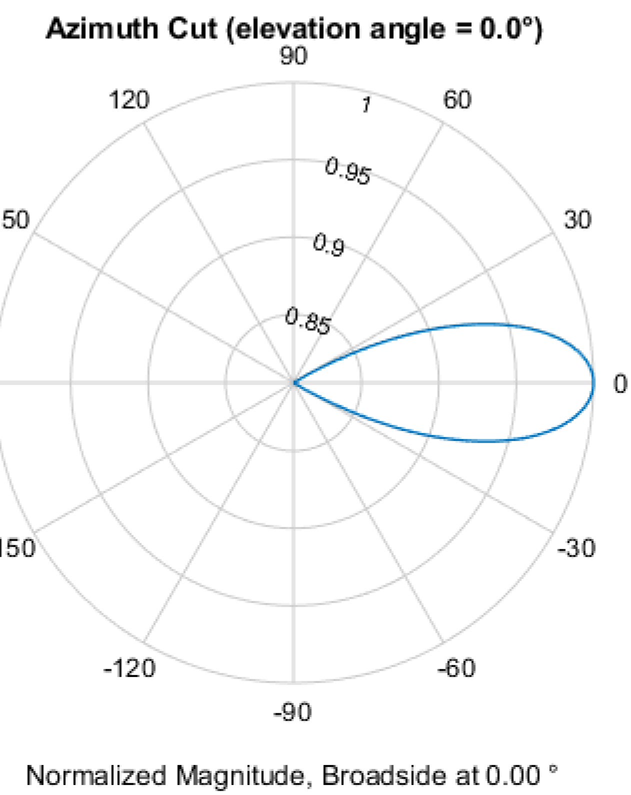
```
    9.3985
    8.6654
    7.3890
```

The maximum directivity is at boresight.

Plot Azimuth-Cut of Cosine Antenna Response Pattern

Construct a cosine antenna element using default parameters. Then, plot the pattern of the field magnitude. Assume the antenna operating frequency is 1 GHz. Restrict the response to the range of azimuth angles from -30 to 30 degrees in 0.1 degree increments. The default elevation angle is 0 degrees.

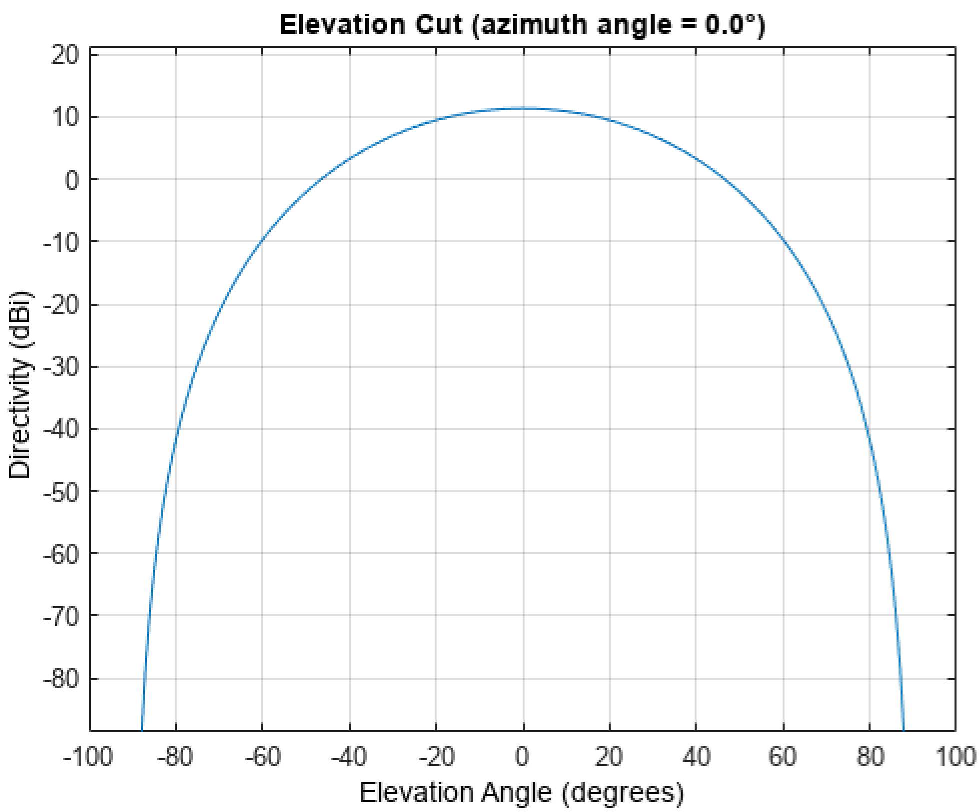
```
antenna = phased.CosineAntennaElement;
fc = 1e9;
pattern(antenna,fc,[-30:0.1:30],0,'Type','efield', ...
    'CoordinateSystem','polar')
```



Plot Directivity of Cosine Antenna

Construct a cosine-pattern antenna. Assume the antenna works between 1 and 2 GHz and its operating frequency is 1.5 GHz. Set the azimuth angle cosine power to 2.5 and the elevation angle cosine power to 3.5. Then, plot an elevation cut of its directivity.

```
antenna = phased.CosineAntennaElement('FrequencyRange', ...
    [1e9 2e9],'CosinePower',[2.5,3.5]);
fc = 1.5e9;
pattern(antenna,fc,0,-90:90,'Type','directivity', ...
    'CoordinateSystem','rectangular')
```

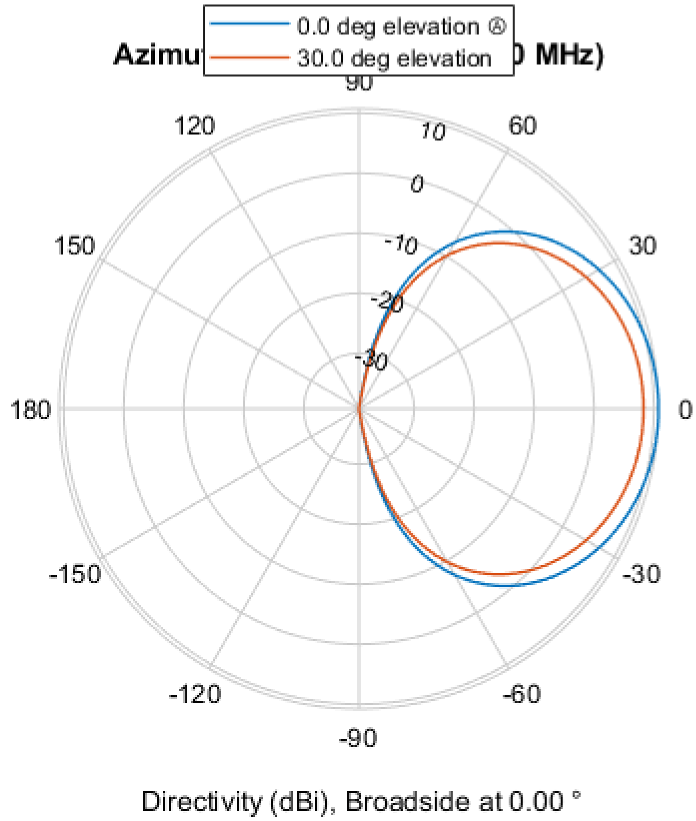


The directivity is maximum at 0 degrees elevation and attains a value of approximately 12 dB.

Limited-Angle Azimuth Pattern of Cosine Antenna

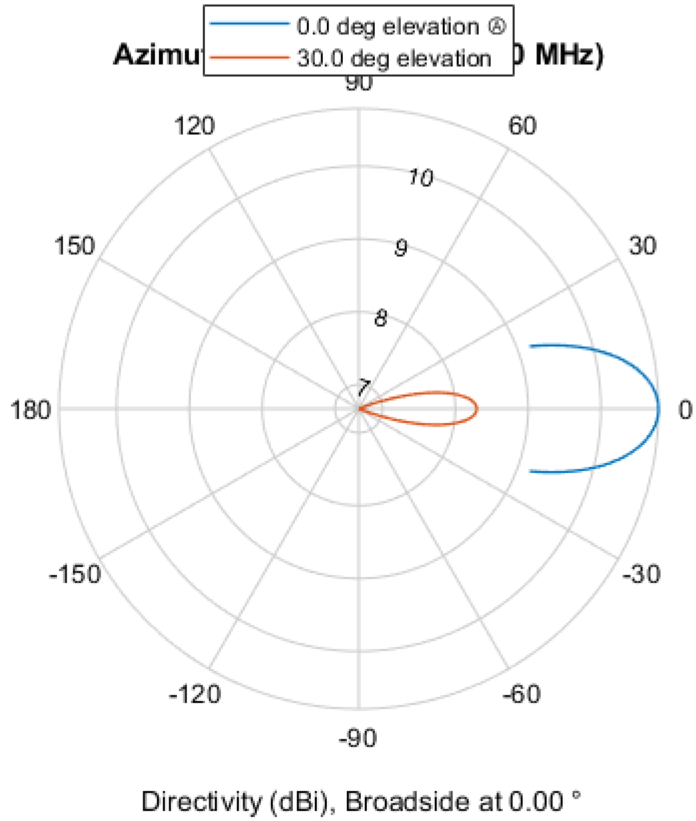
Plot constant-elevation azimuth directivity patterns of a cosine antenna element at 0 degrees and 30 degrees elevation. Assume the operating frequency is 500 MHz.

```
fc = 500e6;
antenna = phased.CosineAntennaElement('FrequencyRange',[100,900]*1e6, ...
    'CosinePower',[3,2]);
patternAzimuth(antenna,fc,[0 30])
```



Plot a limited range of azimuth angles by specifying the `Azimuth` parameter. Note the change in scale.

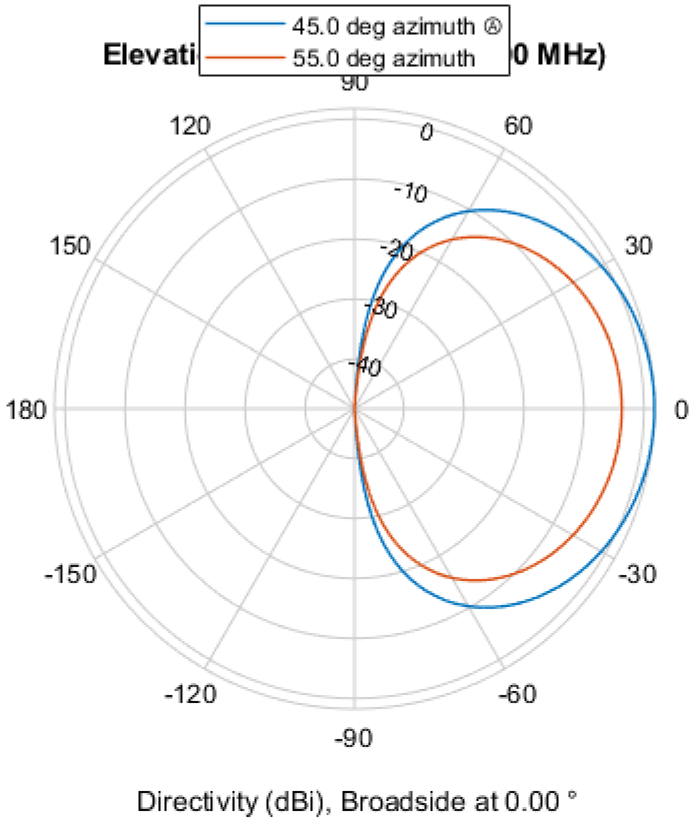
```
patternAzimuth(antenna,fc,[0 30],'Azimuth',-20:20)
```



Limited-Angle Elevation Pattern of Cosine Antenna

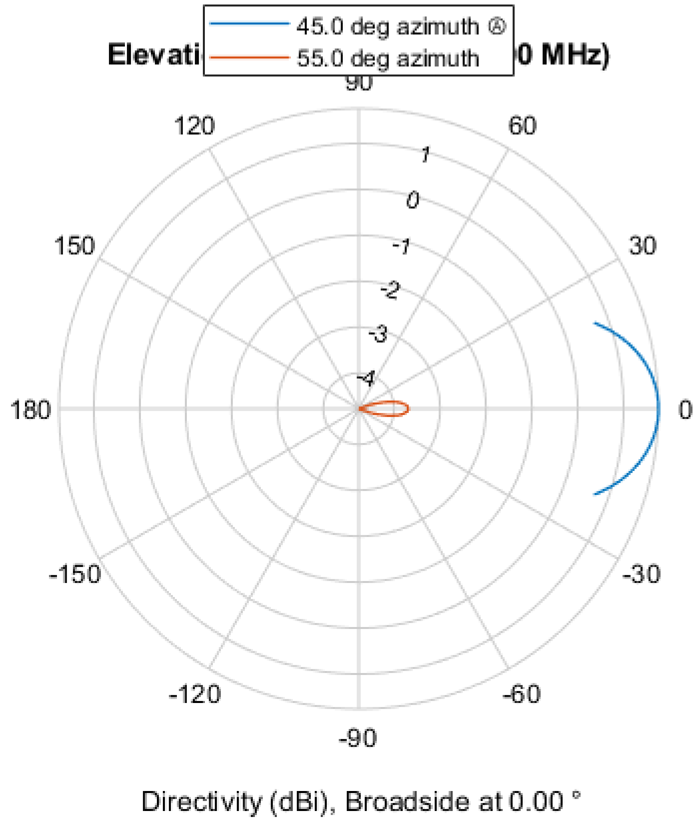
Plot constant-azimuth elevation directivity patterns of a cosine antenna element at 45 and 55 degrees azimuth. Assume the operating frequency is 500 MHz.

```
fc = 500e6;
antenna = phased.CosineAntennaElement('FrequencyRange',[100,900]*1e6, ...
    'CosinePower',[3,2]);
patternElevation(antenna,fc,[45 55])
```



Plot a limited range of elevation angles using the Elevation parameter. Note the change in scale.

```
patternElevation(antenna,fc,[45 55],'Elevation',-20:20)
```



Cosine Antenna Does Not Support Polarization

Create a cosine antenna element using the `phased.CosineAntennaElement` System object™ and show that it does not support polarization.

```
antenna = phased.CosineAntennaElement('FrequencyRange',[1.0,10]*1e9);
isPolarizationCapable(antenna)
```

```
ans = logical
   0
```

The returned value `0` shows that the antenna element does not support polarization.

More About

Cosine Response

The object returns the field response (also called field pattern)

$$f(az, el) = \cos^m(az)\cos^n(el)$$

of the cosine antenna element.

In this expression

- *az* is the azimuth angle.
- *el* is the elevation angle.
- The exponents *m* and *n* are real numbers greater than or equal to zero.

The response is defined for azimuth and elevation angles between –90° and 90°, inclusive, and is always positive. There is no response at the backside of a cosine antenna. The cosine response pattern achieves a maximum value of 1 at 0° azimuth and 0° elevation. Larger exponent values narrow the response pattern of the element and increase the directivity.

The power response (or power pattern) is the squared value of the field response.

$$P(az, el) = \cos^{2m}(az)\cos^{2n}(el)$$

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `pattern`, `patternAzimuth`, and `patternElevation` object functions are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.CrossedDipoleAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ShortDipoleAntennaElement` | `phased.ULA` | `phased.URA` | `phased.UCA` | `phased.ConformalArray` | `phased.CardioidAntennaElement` | `phased.SincAntennaElement` | `phased.GaussianAntennaElement` | `uv2azel` | `phitheta2azel`

phased.CrossedDipoleAntennaElement

Package: `phased`

Crossed-dipole antenna element

Description

The `phased.CrossedDipoleAntennaElement` System object models a crossed-dipole antenna element which is used to generate circularly polarized fields. A crossed-dipole antenna is formed from two orthogonal short-dipole antennas. By default, one dipole lies along *y*-axis and the other along the *z*-axis in the antenna local coordinate system. You can rotate the antenna in the *yz*-plane using the `RotationAngle` property. This antenna object generates right hand or left hand circularly polarized fields, or linearly polarized fields controlled using the `Polarization` property. These fields are pure along the *x*-axis (defined by 0° azimuth and 0° elevation angles).

To compute the response of the antenna element:

1. Create the `phased.CrossedDipoleAntennaElement` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
antenna = phased.CrossedDipoleAntennaElement
antenna = phased.CrossedDipoleAntennaElement(Name,Value)
```

Description

`antenna = phased.CrossedDipoleAntennaElement` creates a crossed-dipole `antenna` with default property values.

`antenna = phased.CrossedDipoleAntennaElement(Name,Value)` creates a crossed-dipole `antenna` with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`FrequencyRange` — Operating frequency range
`[0 1e20]` (default) | nonnegative, real-valued 1-by-2 row vector

Operating frequency range of the antenna, specified as a nonnegative, real-valued, 1-by-2 row vector in the form `[LowerBound HigherBound]`. The antenna element has no response outside the specified frequency range. Units are in Hz.

Data Types: `double`

`RotationAngle` — Crossed-dipole rotation angle
`0` (default) | scalar between -45° and +45°

Crossed-dipole rotation angle, specified as a scalar between -45° and +45°. The rotation angle specifies the angle of rotation of the two dipoles around the *x*-axis. The rotation angle is measured counter-clockwise around the *x*-axis looking towards to origin. A default value of 0° corresponds to the case where one dipole is along the *z*-axis and the other dipole is along the *y*-axis. Units are in degrees.

Data Types: `double`

`Polarization` — Crossed-dipole field polarization
`'RHCP'` (default) | `'LHCP'` | `'Linear'`

Polarization of the field generated by the antenna, specified as `'RHCP'`, `'LHCP'`, or `'Linear'`.

- `'RHCP'` – right hand circularly polarize field. The horizontal field has a 90° phase advance compared to the vertical field.
- `'LHCP'` – left hand circularly polarize field. The horizontal field has a 90° delay compared to the vertical field.
- `'Linear'` – linearly polarized field. The horizontal and vertical fields are in phase.

Example: `'Linear'`

Data Types: `char` | `string`

Usage

Syntax

```
RESP = antenna(FREQ,ANG)
```

Description

`RESP = antenna(FREQ,ANG)` returns the antenna voltage response, `RESP`, at the operating frequencies specified in `FREQ` and in the directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

FREQ — Operating frequency of antenna element
nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

`FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the response is returned as `–Inf`. Element objects use the `FrequencyRange` property, except for `phased.CustomAntennaElement`, which uses the `FrequencyVector` property.

Example: `[1e8 2e6]`

Data Types: `double`

ANG — Azimuth and elevation angles of response directions
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If `ANG` is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of "Azimuth and Elevation Angles".

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

RESP — Antenna voltage response
structure

Voltage response of the antenna, returned as a MATLAB structure with fields `H` and `V`. `H` and `V` contain responses for the horizontal and vertical polarization components of the radiation fields, respectively. Both `H` and `V` are complex-valued, *M*-by-*L* matrices. *M* represents the number of angles specified in `ANG`, and *L* represents the number of frequencies specified in `FREQ`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Compute Crossed-Dipole Antenna Response

Find the response of a crossed-dipole antenna at boresight, 0° azimuth and 0° elevation, and off-boresight at 30° azimuth and 0° elevation. The antenna operates at 250 MHz.

```
antenna = phased.CrossedDipoleAntennaElement('FrequencyRange',[100 900]*1e6);
ang = [0 30;0 0];
fc = 250e6;
resp = antenna(fc,ang);
disp(resp.H)
```

```
   0.0000 - 1.2247i
   0.0000 - 1.0607i
```

```
disp(resp.V)
```

```
   -1.2247
   -1.2247
```

Plot Response of a Crossed-Dipole Antenna

Plot the response patterns of a crossed-dipole antenna used in an L-band radar with a frequency range between 1-2 GHz. First, set up the radar parameters, and obtain the vertical and horizontal polarization responses in five different directions specified by elevation angles of -30, -15, 0, 15 and 30 degrees, all at 0 degrees azimuth angle. The responses are computed at an operating frequency of 1.5 GHz.

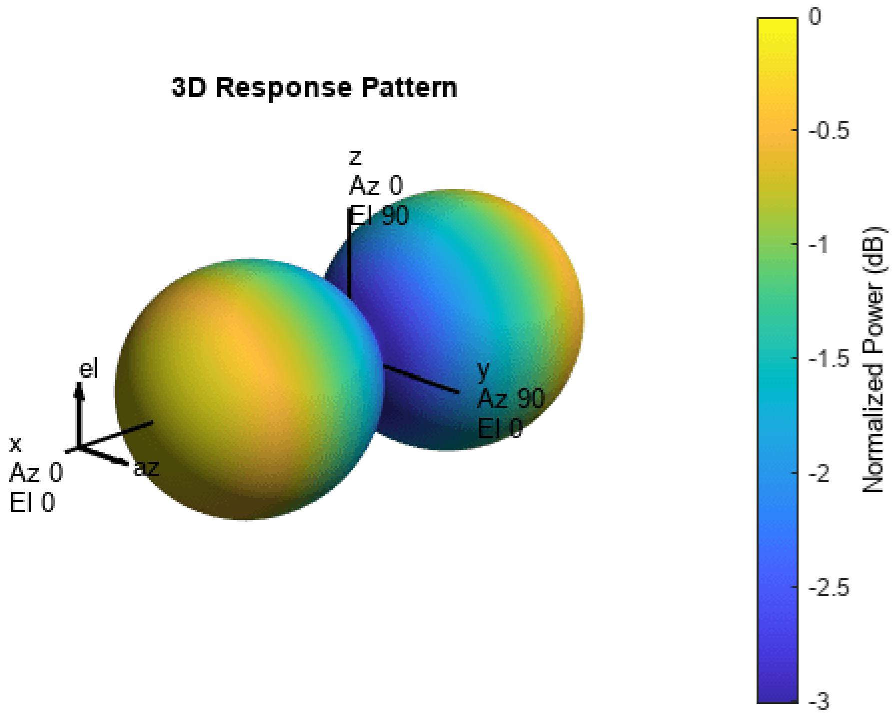
```
antenna = phased.CrossedDipoleAntennaElement('FrequencyRange',[1,2]*1e9);
fc = 1.5e9;
resp = antenna(fc,[0,0,0,0,0;-30,-15,0,15,30]);
[resp.V, resp.H]
```

```
ans = 5×2 complex
```

```
  -1.0607 + 0.0000i   0.0000 - 1.2247i
  -1.1830 + 0.0000i   0.0000 - 1.2247i
  -1.2247 + 0.0000i   0.0000 - 1.2247i
  -1.1830 + 0.0000i   0.0000 - 1.2247i
  -1.0607 + 0.0000i   0.0000 - 1.2247i
```

Next, draw a 3-D plot of the combined polarization response.

```
pattern(antenna,fc,-180:180,-90:90,'CoordinateSystem','polar', ...
    'Type','powerdb','Polarization','combined')
```



Directivity of Crossed-Dipole Antenna Element

Compute the directivity of a crossed-dipole antenna element in several different directions.

Create a crossed-dipole antenna element System object™.

```
antenna = phased.CrossedDipoleAntennaElement;
```

Set the angles of interest to be at zero-degrees constant elevation angle. The seven azimuth angles are centered around boresight (zero degrees azimuth and zero degrees elevation). Set the desired frequency to 1 GHz.

```
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
freq = 1e9;
```

Compute the directivity along the constant elevation cut.

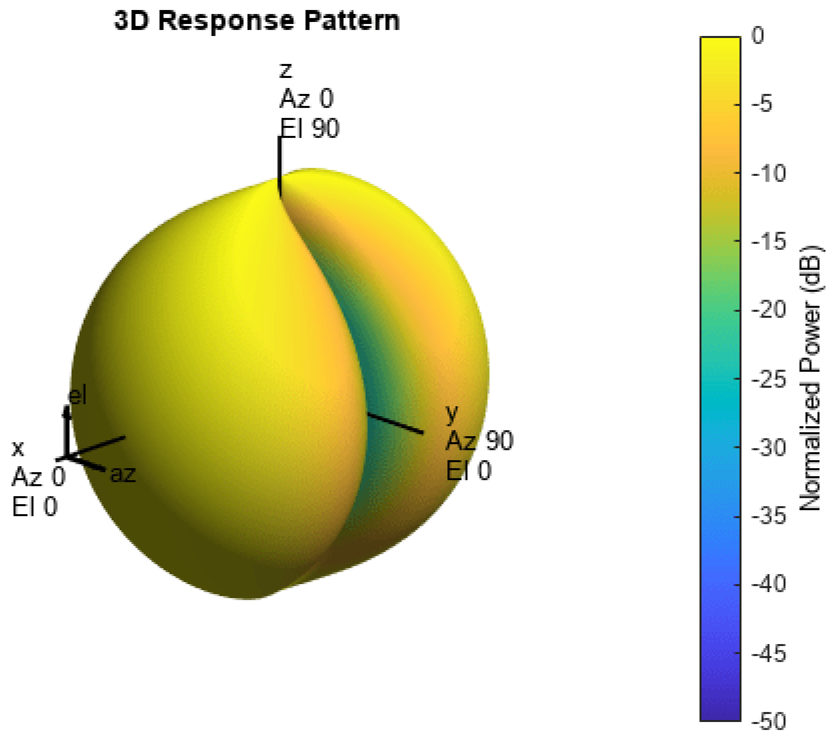
```
d = directivity(antenna,freq,ang)
```

```
d = 7×1

    1.1811
    1.4992
    1.6950
    1.7610
    1.6950
    1.4992
    1.1811
```

Plot 3-D Polar Patterns of Crossed-Dipole Antenna

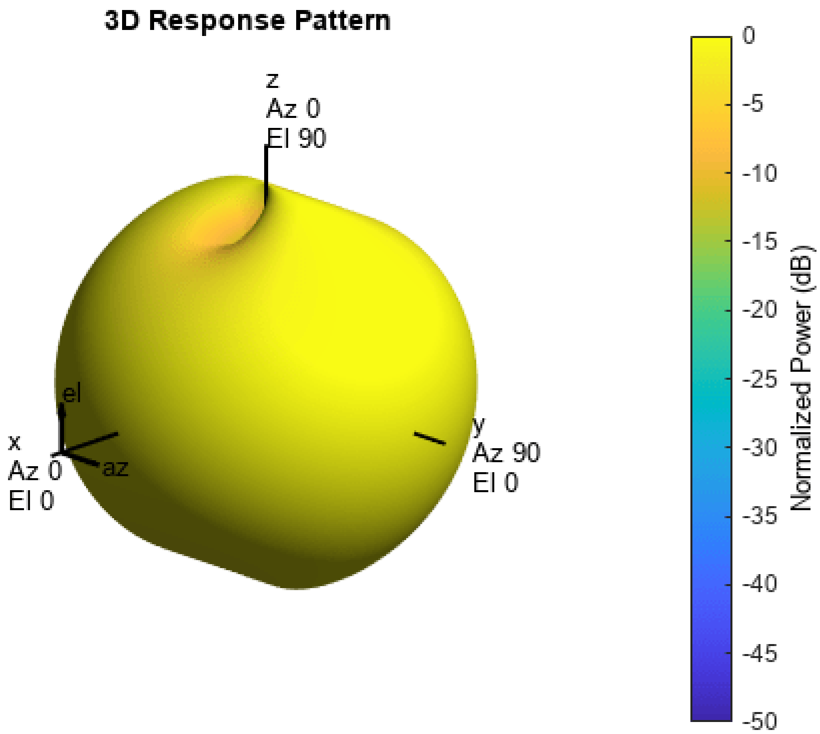
Construct a crossed-dipole antenna element that operates in the frequency range from 100 MHz to 1.5 GHz. Then, plot the 3-D polar power pattern for the horizontal polarization component. Assume the antenna operates at 1 GHz.

```
antenna = phased.CrossedDipoleAntennaElement('FrequencyRange',[100 1500]*1e6);
fc = 1e9;
pattern(antenna,fc,-180:180,-90:90,'Type','powerdb', ...
    'CoordinateSystem','polar','Polarization','H')
```



Next, plot the vertical polarization component.

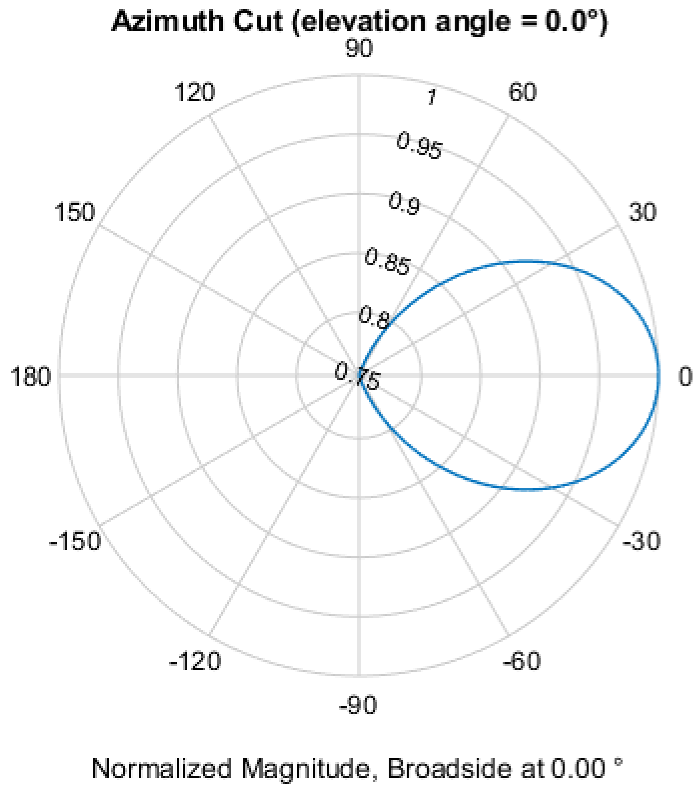
```
pattern(antenna,fc,-180:180,-90:90,'Type','powerdb', ...
    'CoordinateSystem','polar','Polarization','V')
```



Plot Crossed-Dipole Antenna Pattern at Constant Elevation

Construct a crossed-dipole antenna element. Then, plot the pattern of the horizontal component of the field magnitude at an elevation angle of 0 degrees. Assume the antenna operating frequency is 1 GHz. Restrict the response to the range of azimuth angles from -70 to 70 degrees in 0.1 degree increments.

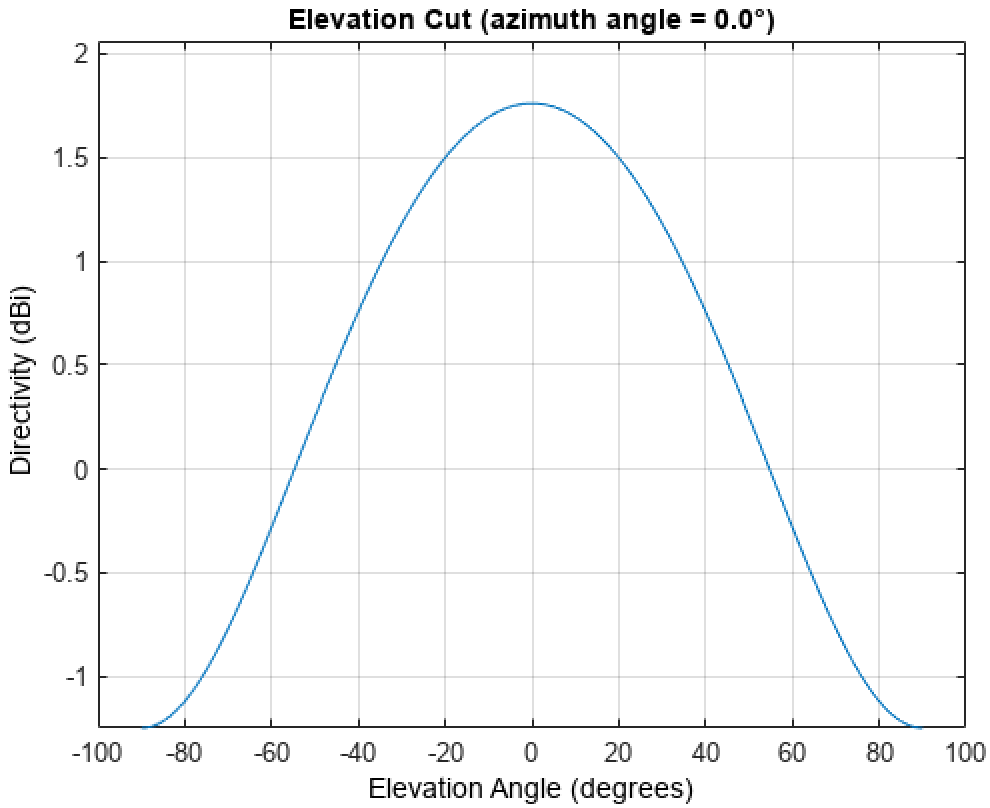
```
antenna = phased.CrossedDipoleAntennaElement('FrequencyRange',[0.5 1.5]*1e9);
fc = 1e9;
pattern(antenna,fc,-70:0.1:70,0,'Type','efield', ...
    'CoordinateSystem','polar','Polarization','combined')
```



Plot Directivity of Crossed-Dipole Antenna

Create a crossed-dipole antenna. Assume the antenna works between 1 and 2 GHz and its operating frequency is 1.5 GHz. Then, plot the directivity at a constant azimuth of 0˚.

```
antenna = phased.CrossedDipoleAntennaElement('FrequencyRange',[1e9 2e9]);
fc = 1.5e9;
pattern(antenna,fc,0,-90:90,'Type','directivity', ...
    'CoordinateSystem','rectangular')
```

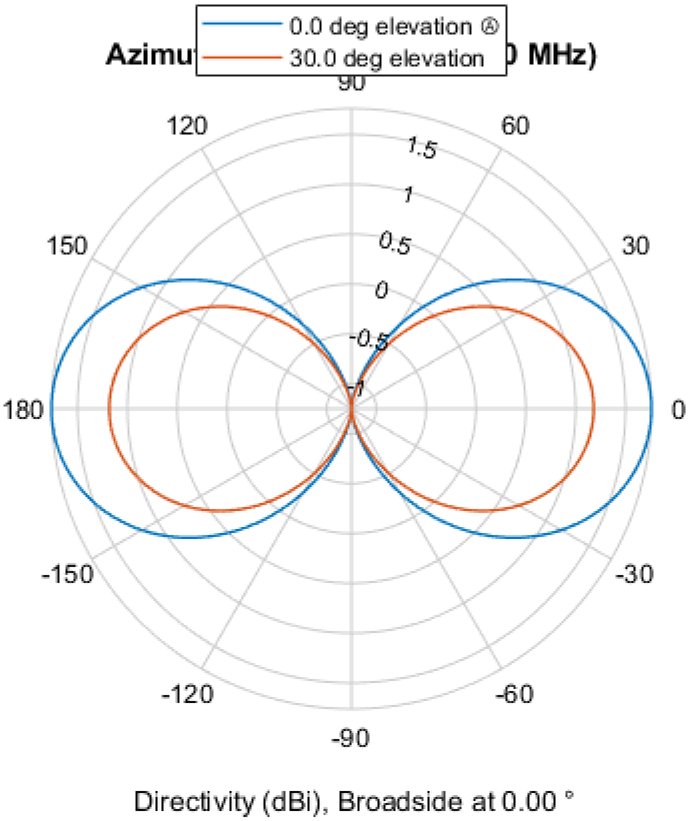


The directivity is maximum at 0˚ elevation and attains a value of approximately 1.75 dB.

Plot Azimuth Pattern of Crossed-Dipole Antenna Element

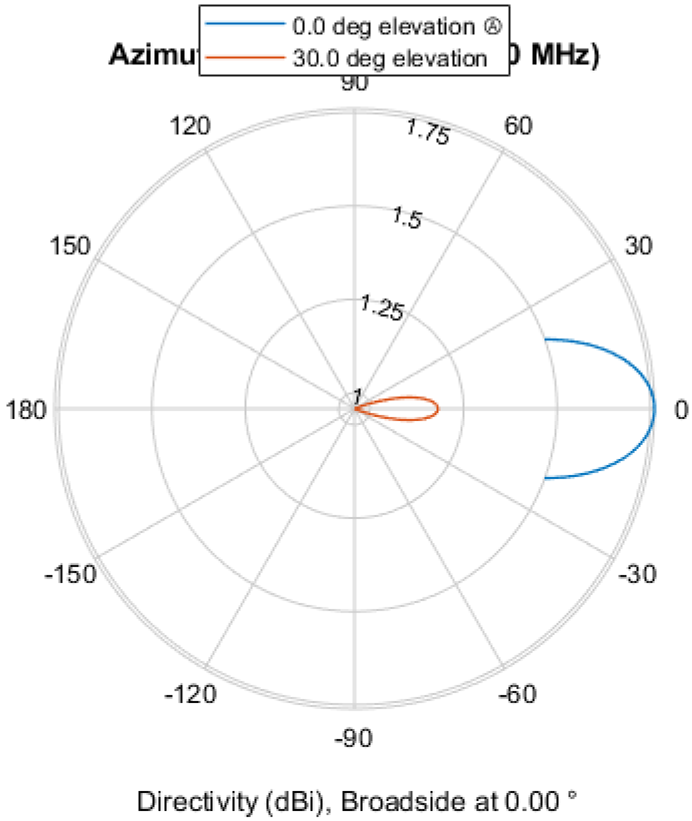
Plot the azimuth directivity pattern of a crossed-dipole antenna at two different elevations: 0˚ and30˚. Assume the operating frequency is 500 MHz.

```
fc = 500e6;
antenna = phased.CrossedDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
patternAzimuth(antenna,fc,[0 30])
```



Plot a limited range of azimuth angles using the `Azimuth` parameter. Notice the change in scale.

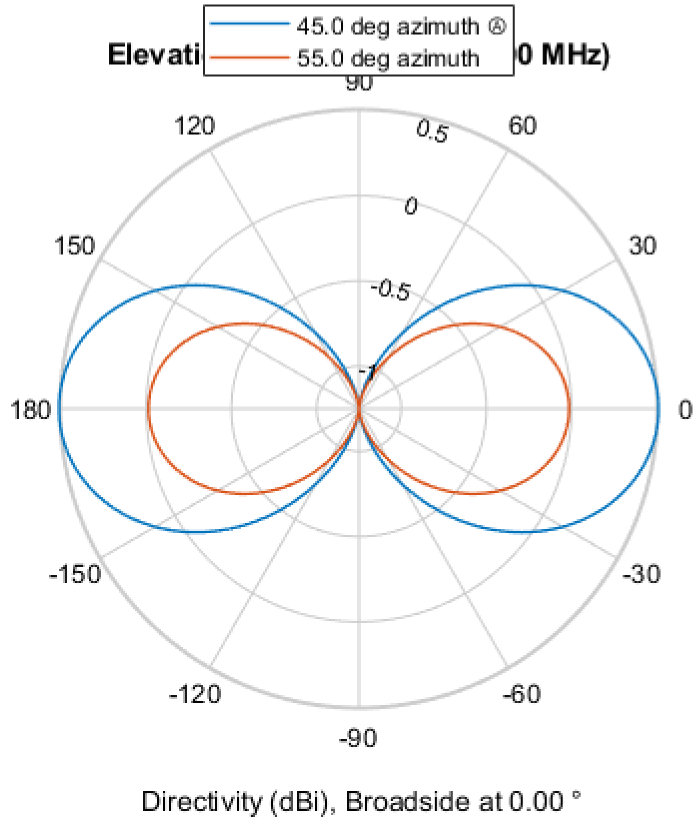
```
patternAzimuth(antenna,fc,[0 30],'Azimuth',[-20:20])
```



Plot Elevation Pattern of Crossed-Dipole Antenna Element

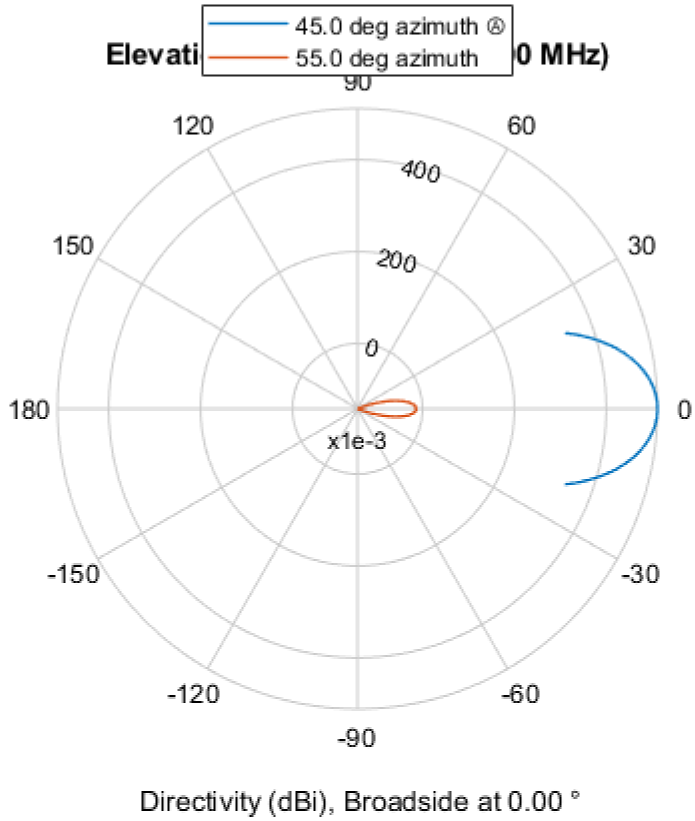
Plot the elevation directivity pattern of a crossed-dipole antenna at two different azimuths: 45° and 55°. Assume the operating frequency is 500 MHz.

```
fc = 500e6;
sCD = phased.CrossedDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
patternElevation(sCD,fc,[45 55])
```



Plot a reduced range of elevation angles using the `Elevation` parameter. Notice the change in scale.

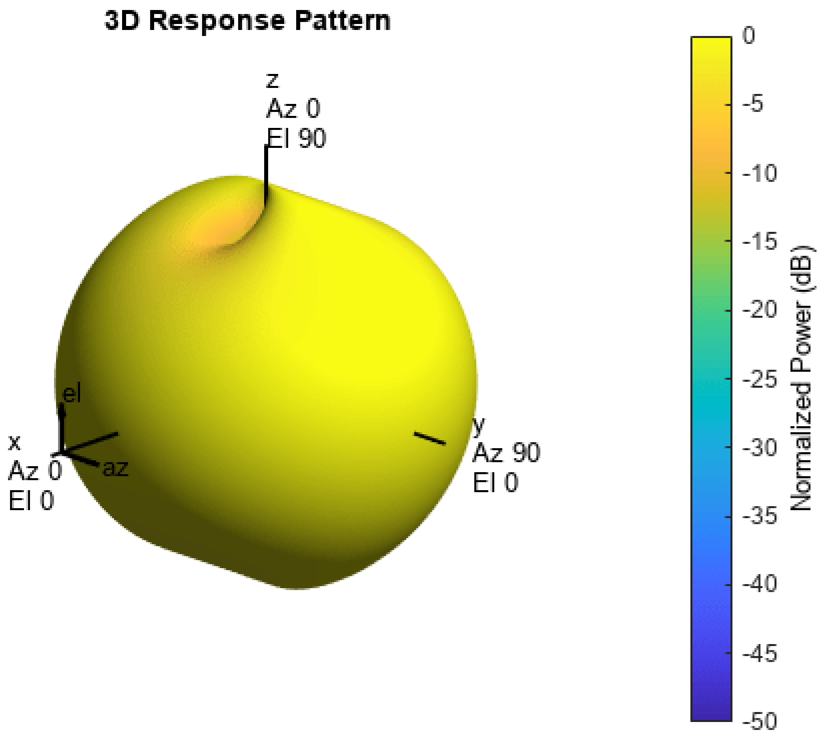
```
patternElevation(sCD,fc,[45 55],'Elevation',-20:20)
```



Vertical and Horizontal Responses of Crossed-Dipole Antenna

This example shows how to create a crossed-dipole antenna operating between 100 and 900 MHz and then how to plot its vertical and horizontal polarization response at 250 MHz in the form of a 3-D polar plot.

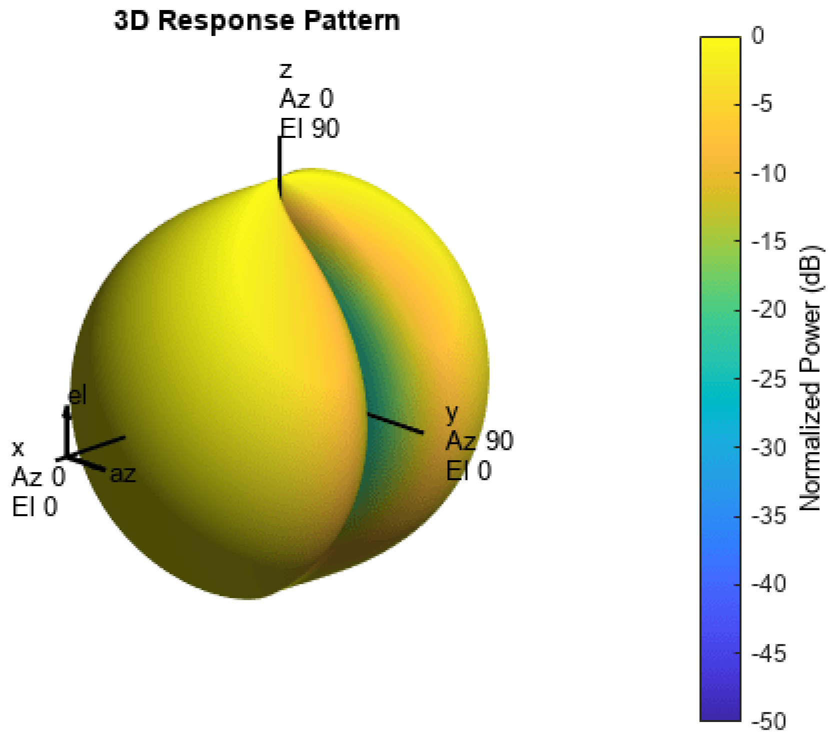
```
antenna = phased.CrossedDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6);
pattern(antenna,250e6,-180:180,-90:90,'CoordinateSystem','polar','Polarization','V', ...
    'Type','powerdb')
```



The antenna pattern of the vertical-polarization component is almost isotropic and has a maximum at 0˚ elevation and 0˚ azimuth, as shown in the figure above.

Plot the antenna's horizontal polarization response. The pattern of the horizontal polarization response also has a maximum at 0˚ elevation and 0˚ azimuth but no response at ±90˚ azimuth.

```
pattern(antenna,250e6,-180:180,-90:90,'CoordinateSystem','polar','Polarization','H', ...
    'Type','powerdb')
```



Crossed-Dipole Antenna Supports Polarization

Show that the `phased.CrossedDipoleAntennaElement` antenna element supports polarization.

```
antenna = phased.CrossedDipoleAntennaElement;
isPolarizationCapable(antenna)
```

```
ans = logical
   1
```

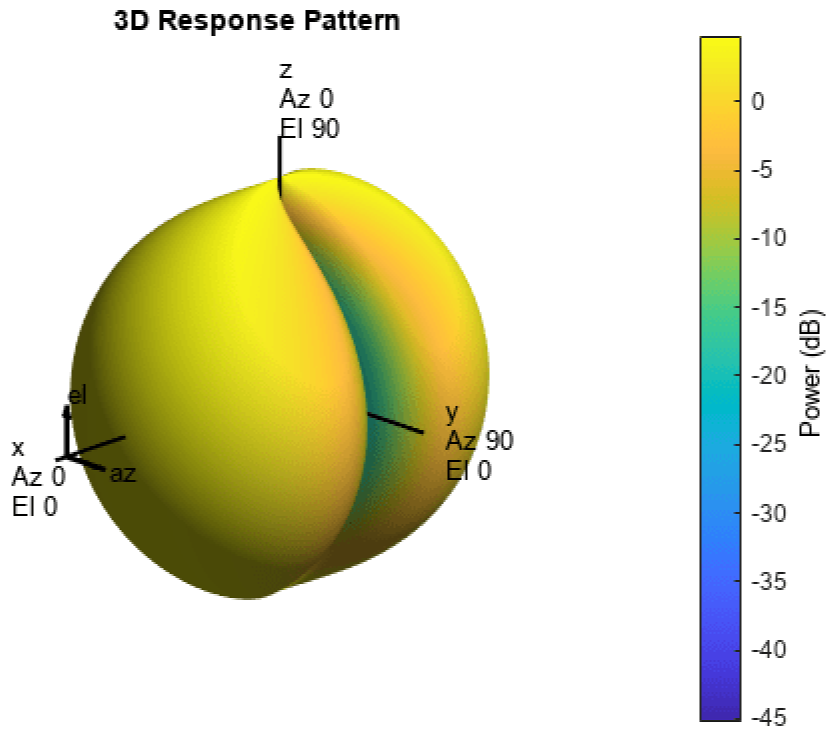
The returned value of `1` shows that the crossed-dipole antenna element supports polarization.

Plot 3-D Polar Patterns of Rotated Crossed-Dipole Antenna

Construct a crossed-dipole antenna element designed to operate in the frequency range from 100 MHz to 1.5 GHz. Assume the polarization is linear. Rotate the antenna by -45 degrees. Plot the 3-D polar power pattern for the horizontal and vertical polarization components at 1 GHz.

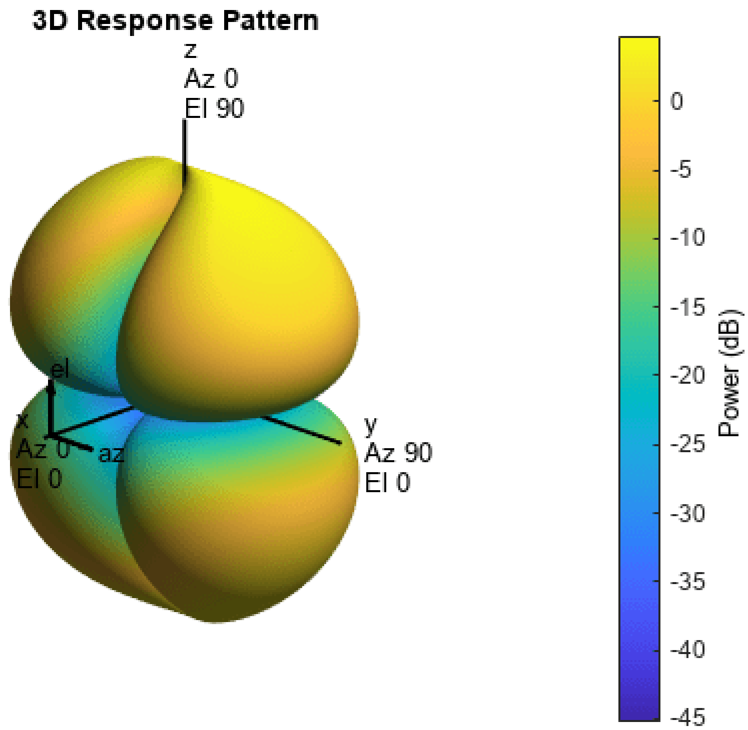
```
antenna = phased.CrossedDipoleAntennaElement('FrequencyRange',[100 1500]*1e6, ...
   'RotationAngle',-45.0,'Polarization','Linear');
fc = 1e9;
```

```
pattern(antenna,fc,-180:180,-90:90,'Type','powerdb','Normalize',false, ...
    'CoordinateSystem','polar','Polarization','H')
```



Next, plot the vertical polarization component.

```
pattern(antenna,fc,-180:180,-90:90,'Type','powerdb','Normalize',false, ...
    'CoordinateSystem','polar','Polarization','V')
```



Algorithms

The total response of a crossed-dipole antenna element is a combination of its frequency response and spatial response. `phased.CrossedDipoleAntennaElement` calculates both responses using nearest neighbor interpolation, and then multiplies the responses to form the total response.

Version History

Introduced in R2013a

References

[1] Mott, H., *Antennas for Radar and Communications*, John Wiley & Sons, 1992.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `pattern`, `patternAzimuth`, and `patternElevation` object functions are not supported.

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also
`phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ShortDipoleAntennaElement` | `phased.ULA` | `phased.URA` | `phased.UCA` | `phased.ConformalArray` | `uv2azelpat` | `phitheta2azelpat` | `uv2azel` | `phitheta2azel`

phased.CustomAntennaElement

Package: `phased`

Custom antenna element

Description

The `phased.CustomAntennaElement` System object models an antenna element with a custom spatial response pattern. The response pattern can be defined for polarized or non-polarized fields.

To create a custom antenna element:

1. Create the `phased.CustomAntennaElement` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
antenna = phased.CustomAntennaElement
antenna = phased.CustomAntennaElement(Name,Value)
```

Description

`antenna = phased.CustomAntennaElement` creates a System object, `antenna`, with default property values. The default response pattern is spatially isotropic.

`antenna = phased.CustomAntennaElement(Name,Value)` creates a custom antenna object, `antenna`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). For example, the output response of the object depends on whether polarization is set or not.

- To create a nonpolarized response pattern, set the `SpecifyPolarizationPattern` property to `false` (default). Then, use the `MagnitudePattern` and `PhasePattern` properties to define the response pattern.
- To create a polarized response pattern, set the `SpecifyPolarizationPattern` property to `true`. Then, use any or all of the `HorizontalMagnitudePattern`, `HorizontalPhasePattern`, `VerticalMagnitudePattern`, and `VerticalPhasePattern` properties to define the response pattern.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`FrequencyVector` — Response and pattern frequency vector
`[0 1e20]` (default) | 1-by-*L* row vector

Frequencies at which the frequency response and antenna patterns are to be returned, specified as a 1-by-*L* row vector. The elements of the vector must be in increasing order. The antenna element has no response outside the frequency range specified by the minimum and maximum elements of the frequency vector. Units are in Hz.

Example: `[200:50:300]*1e6`

Data Types: `double`

`FrequencyResponse` — Frequency responses of antenna element
`[0 0]` (default) | real-valued 1-by-*L* vector

Frequency responses at the frequencies defined in `FrequencyVector` property, specified as a 1-by-*L* row vector. *L* equals the length of the vector specified in the `FrequencyVector` property. Units are in dB.

Example: `[0 6 0]`

Data Types: `double`

`PatternCoordinateSystem` — Coordinate system of custom antenna pattern
`'az-el'` (default) | `'phi-theta'`

Coordinate system of custom antenna pattern, specified `'az-el'` or `'phi-theta'`. When you specify `'az-el'`, use the `AzimuthAngles` and `ElevationAngles` properties to specify the pattern coordinates system. When you specify `'phi-theta'`, use the `PhiAngles` and `ThetaAngles` properties to specify the pattern coordinates system.

Data Types: `char`

`AzimuthAngles` — Azimuth angles
`[-180:180]` (default) | real-valued length-*P* vector

Specify the azimuth angles as a length-*P* vector. These angles are the azimuth angles where the custom radiation pattern is specified. *P* must be greater than 2. The azimuth angles must lie between –180° and 180° and be in strictly increasing order. Units are in degrees.

Example: `[30 40 50]`

Dependencies

To enable this property, set the `PatternCoordinateSystem` property to `'az-el'`.

Data Types: `double`

`ElevationAngles` — Elevation angles
`[-90:90]` (default) | real-valued length-*Q* vector

Specify the elevation angles as a length-*Q* vector. These angles are the elevation angles where the custom radiation pattern is specified. *Q* must be greater than 2. The elevation angles must lie between –90° and 90° and be in strictly increasing order. Units are in degrees.

Example: `[-30 0 +30]`

Dependencies

To enable this property, set the `PatternCoordinateSystem` property to `'az-el'`.

Data Types: `double`

`PhiAngles` — Phi angles in phi-theta coordinates system
`0:360` (default) | real-valued *P*-length vector

Phi angles used to represent the element response pattern in phi-theta coordinates, specified as a real-valued *P*-length vector. Phi angles lie between 0° and 360°. *P* must be greater than 2. Look here for definitions of “Phi and Theta Angles”.

Example: `[90:180]`

Dependencies

To enable this property, set the `PatternCoordinateSystem` property to `'phi-theta'`.

Data Types: `double`

`ThetaAngles` — Theta angles in phi-theta coordinate system
`0:180` (default) | real-valued *Q*-length vector

Theta angles used to represent the element response pattern in phi-theta coordinates, specified as a real-valued *Q*-length vector. The theta angle lies between 0° and 180°. Look here for definitions of “Phi and Theta Angles”. *Q* must be greater than 2.

Example: `[40:80]`

Dependencies

To enable this property, set the `PatternCoordinateSystem` property to `'phi-theta'`.

Data Types: `double`

`SpecifyPolarizationPattern` — Polarized array response
`false` (default) | `true`

Polarized array response, specified as `false` or `true`.

- When the `SpecifyPolarizationPattern` property is set to `false`, the antenna element transmits or receives non-polarized radiation. In this case, use the `MagnitudePattern` property to set the antenna response pattern.
- When the `SpecifyPolarizationPattern` property is set to `true`, the antenna element transmits or receives polarized radiation. In this case, use the `HorizontalMagnitudePattern` and `HorizontalPhasePattern` properties to set the horizontal polarization response pattern and the `VerticalMagnitudePattern` and `VerticalPhasePattern` properties to set the vertical polarization response pattern.

Data Types: `logical`

`MagnitudePattern` — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

The magnitude of the combined polarization antenna radiation, pattern specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array. This property is used only when the `SpecifyPolarizationPattern` property is set to `false`. Magnitude units are in dB.

- If the value of this property is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

If the pattern contains a `NaN` at any azimuth and elevation direction, it is converted to `-Inf`, indicating zero response in that direction. The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern must contain azimuth angles in the range `[–180,180]` degrees. Set the range of elevation angles to `[–90,90]` degrees.

Data Types: `double`

`PhasePattern` — Phase of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

The phase of the combined polarization antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array. This property is used only when the `SpecifyPolarizationPattern` property is set to `false`. Units are in degrees.

- If the value of this property is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern must contain azimuth angles in the range *[–180°,180°]*. Set the range of elevation angles to *[–90°,90°]*.

Data Types: `double`

`HorizontalMagnitudePattern` — Magnitude of horizontal polarization component of antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

The magnitude of the horizontal polarization component of the antenna radiation pattern, specified as a real-valued *Q*-by-*P* matrix or real-valued a *Q*-by-*P*-by-*L* array. Magnitude units are in dB.

- If the value of this property is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

If the magnitude pattern contains a `NaN` at any azimuth and elevation direction, it is converted to `-Inf`, indicating zero response in that direction. The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern must contain azimuth angles in the range `[–180,180]°` and elevation angles in the range `[–90,90]°`.

Dependencies

To enable this property, set the `SpecifyPolarizationPattern` property to `true`.

Data Types: `double`

`HorizontalPhasePattern` — Phase of horizontal polarization component of antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

The phase of the horizontal polarization component of the antenna radiation pattern, specified as a real-valued *Q*-by-*P* matrix or a real-valued *Q*-by-*P*-by-*L* array. This property is used only when the `SpecifyPolarizationPattern` property is set to `true`. Phase units are in degrees.

- If the value of this property is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern must contain azimuth angles in the range `[–180,180]°` and elevation angles in the range `[–90,90]°`.

Dependencies

To enable this property, set the `SpecifyPolarizationPattern` property to `true`.

Data Types: `double`

`VerticalMagnitudePattern` — Magnitude of vertical polarization component of antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

The magnitude of the vertical polarization component of the antenna radiation pattern specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array. This property is used only when the `SpecifyPolarizationPattern` property is set to `true`. Magnitude units are in dB.

- If the value of this property is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

If the pattern contains a `NaN` at any azimuth and elevation direction, it is converted to `-Inf`, indicating zero response in that direction. The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern must contain azimuth angles in the range`[–180,180]°` and elevation angles in the range `[–90,90]°`.

Dependencies

To enable this property, set the `SpecifyPolarizationPattern` property to `true`.

Data Types: `double`

`VerticalPhasePattern` — Phase of vertical polarization component of antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

The phase of the vertical polarization component of the antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array. This property is used only when the `SpecifyPolarizationPattern` property is set to `true`. Phase units are in degrees.

- If the value of this property is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern must contain azimuth angles in the range `[–180,180]°` and elevation angles in the range `[–90,90]°`.

Dependencies

To enable this property, set the `SpecifyPolarizationPattern` property to `true`.

Data Types: `double`

`MatchArrayNormal` — Match element normal to array normal
`true` (default) | `false`

Set this property to `true` to align the antenna element to an array normal. The antenna pattern is rotated so that the *x*-axis of the element coordinate system points along the array normal. This property is used only when the antenna element belongs to an array. Use the property in conjunction with the `ArrayNormal` property of the `phased.URA` and `phased.UCA` System objects. Set this property to `false` to use the element pattern without rotation. The default value is .

Data Types: `logical`

Usage

Syntax

```
RESP = antenna(FREQ,ANG)
```

Description

`RESP = antenna(FREQ,ANG)` returns the antenna’s voltage response `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`. The form of `RESP` depends upon whether the antenna element supports polarization as determined by the `SpecifyPolarizationPattern` property. If `SpecifyPolarizationPattern` is set to `false`, `RESP` is an *M*-by-*L* matrix containing the antenna response at the *M* angles specified in `ANG` and at the *L* frequencies specified in `FREQ`. If `SpecifyPolarizationPattern` is set to `true`, `RESP` is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`, representing the antenna's response in horizontal and vertical polarization, respectively. Each field is an *M*-by-*L* matrix containing the antenna response at the *M* angles specified in `ANG` and at the *L* frequencies specified in `FREQ`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

FREQ — Operating frequency of antenna element
nonnegative scalar | nonnegative, real-valued 1-by-L row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-L row vector. Frequency units are in Hz.

`FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the response is returned as `–Inf`. Element objects use the `FrequencyRange` property, except for `phased.CustomAntennaElement`, which uses the `FrequencyVector` property.

Example: `[1e8 2e6]`

Data Types: `double`

ANG — Azimuth and elevation angles of response directions
real-valued 1-by-M row vector | real-valued 2-by-M matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-M row vector or a real-valued 2-by-M matrix, where M is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If `ANG` is a 1-by-M vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the x-axis and the projection of the direction vector onto the xy-plane. This angle is positive when measured from the x-axis toward the y-axis. The elevation angle is the angle between the direction vector and xy-plane. This angle is positive when measured toward the z-axis. See the definition of “Azimuth and Elevation Angles”.

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

RESP — Voltage response of antenna
complex-valued M-by-L matrix

Voltage response of antenna element, returned as a complex-valued M-by-L matrix. In this matrix, M represents the number of angles specified in `ANG` and L represents the number of frequencies specified in `FREQ`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Response and Directivity of Custom Antenna

Create a user-defined antenna with a cosine pattern. Then, plot an elevation cut of the antenna's power response.

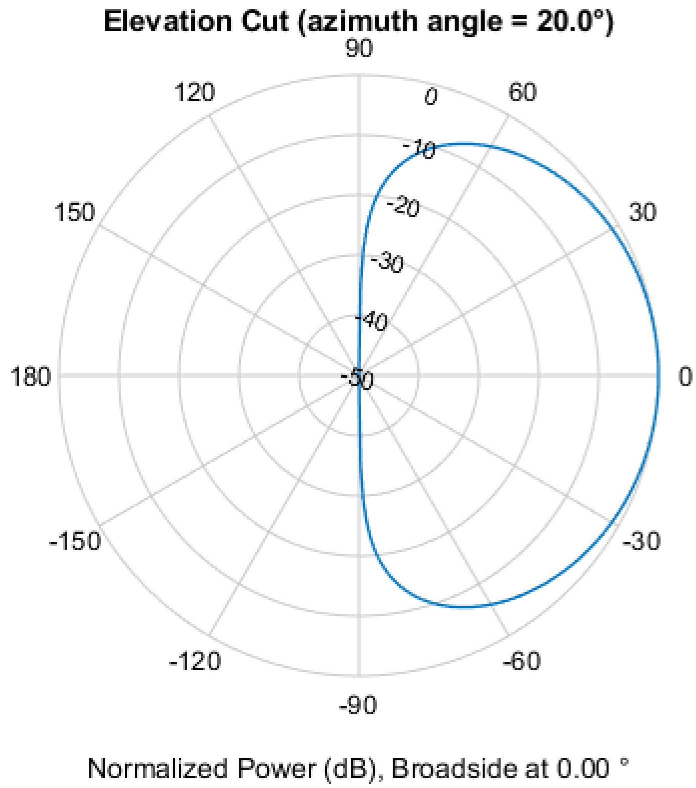
The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna operates at 1 GHz. Obtain the response at 20° azimuth and 30° elevation.

```
fc = 1e9;
azang = -180:180;
elang = -90:90;
magpattern = mag2db(repmat(cosd(elang)',1,numel(azang)));
phasepattern = zeros(size(magpattern));
antenna = phased.CustomAntennaElement('AzimuthAngles',azang, ...
    'ElevationAngles',elang,'MagnitudePattern',magpattern, ...
    'PhasePattern',phasepattern);
resp = antenna(fc,[20;30])
```

```
resp = 0.8660
```

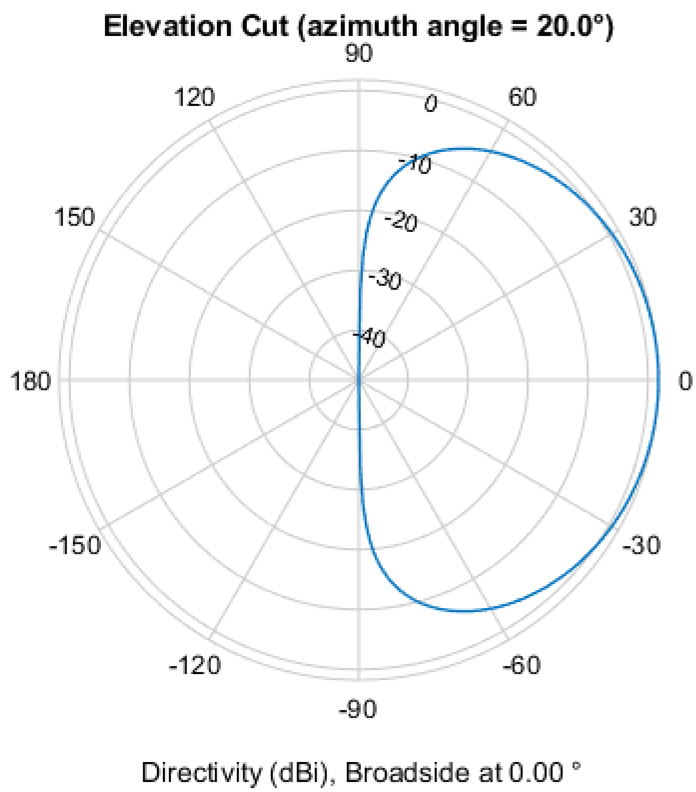
Plot an elevation cut of the power response.

```
pattern(antenna,fc,20,-90:90,'CoordinateSystem','polar','Type','powerdb')
```



Plot an elevation cut of the directivity.

```
pattern(antenna,fc,20,-90:90,'CoordinateSystem','polar','Type','directivity')
```



Antenna Radiation Pattern in U-V Coordinates

Define a custom antenna in *u*-*v* space. Then, calculate and plot the response.

Define the radiation pattern (in dB) of an antenna in terms of *u* and *v* coordinates within the unit circle.

```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Create an antenna with this radiation pattern. Convert *u*-*v* coordinates to azimuth and elevation coordinates.

```
[pat_azel,az,el] = uv2azelpat(pat_uv,u,v);
array = phased.CustomAntennaElement('AzimuthAngles',az,'ElevationAngles',el, ...
    'MagnitudePattern',mag2db(pat_azel),'PhasePattern',45*ones(size(pat_azel)));
```

Calculate the response in the direction $u = 0.5$, $v = 0$. Assume the antenna operates at 1 GHz. The output of the step method is in linear units.

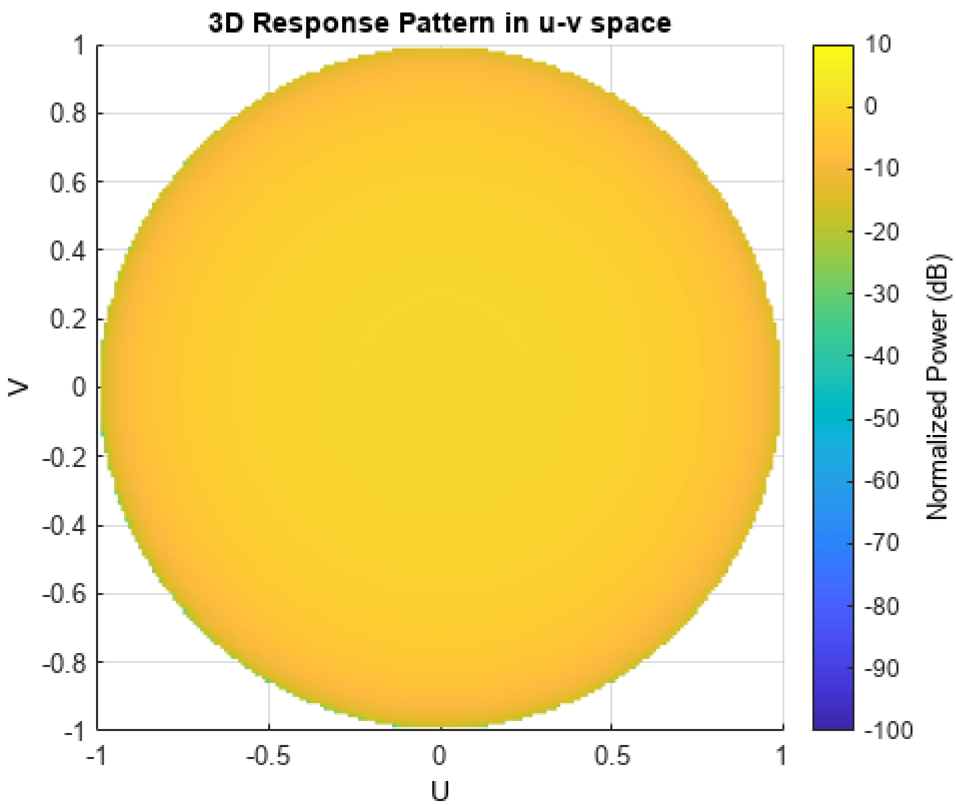
```
dir_uv = [0.5;0];
dir_azel = uv2azel(dir_uv);
```

```
fc = 1e9;
resp = array(fc,dir_azel)
```

```
resp = 0.6124 + 0.6124i
```

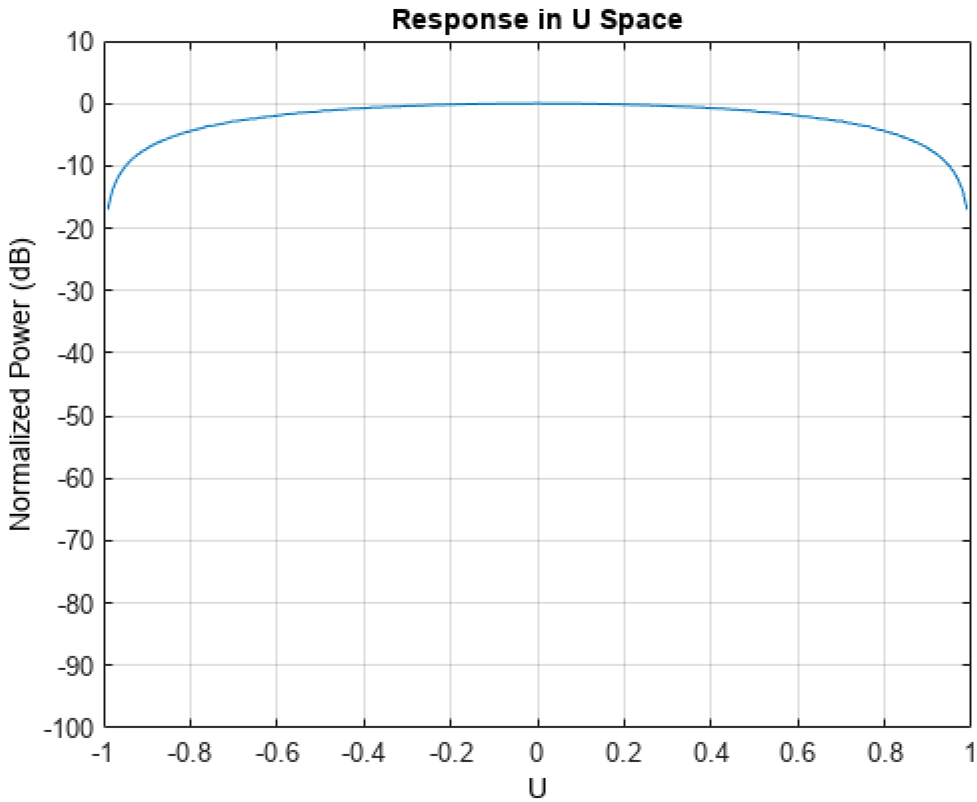
Plot the 3D response in *u*-*v* coordinates.

```
pattern(array,fc,[-1:.01:1],[-1:.01:1],'CoordinateSystem','uv','Type','powerdb')
```



Display the antenna response as a line plot in *u*-*v* coordinates.

```
pattern(array,fc,[-1:.01:1],0,'CoordinateSystem','uv','Type','powerdb')
```



Polarized Antenna Radiation Patterns

Model a short dipole antenna oriented along the *x*-axis of the local antenna coordinate system. For this type of antenna, the horizontal and vertical components of the electric field are given by $E_H = \frac{j\omega\mu IL}{4\pi r}\sin(\mathrm{az})$ and $E_V = -\frac{j\omega\mu IL}{4\pi r}\sin(\mathrm{el})\cos(\mathrm{az})$.

Specify a normalized radiation pattern of a short dipole antenna terms of azimuth, *az*, and elevation, *el*, coordinates. The vertical and horizontal radiation patterns are normalized to a maximum of unity.

```
az = [-180:180];
el = [-90:90];
[az_grid,el_grid] = meshgrid(az,el);
horz_pat_azel = ...
    mag2db(abs(sind(az_grid)));
vert_pat_azel = ...
    mag2db(abs(sind(el_grid).*cosd(az_grid)));
```

Set up the antenna. Specify the `SpecifyPolarizationPattern` property to produce polarized radiation. In addition, use the `HorizontalMagnitudePattern` and `VerticalMagnitudePattern` properties to specify the pattern magnitude values. The `HorizontalPhasePattern` and `VerticalPhasePattern` properties take default values of zero.

```
antenna = phased.CustomAntennaElement(...
    'AzimuthAngles',az,'ElevationAngles',el,...
```

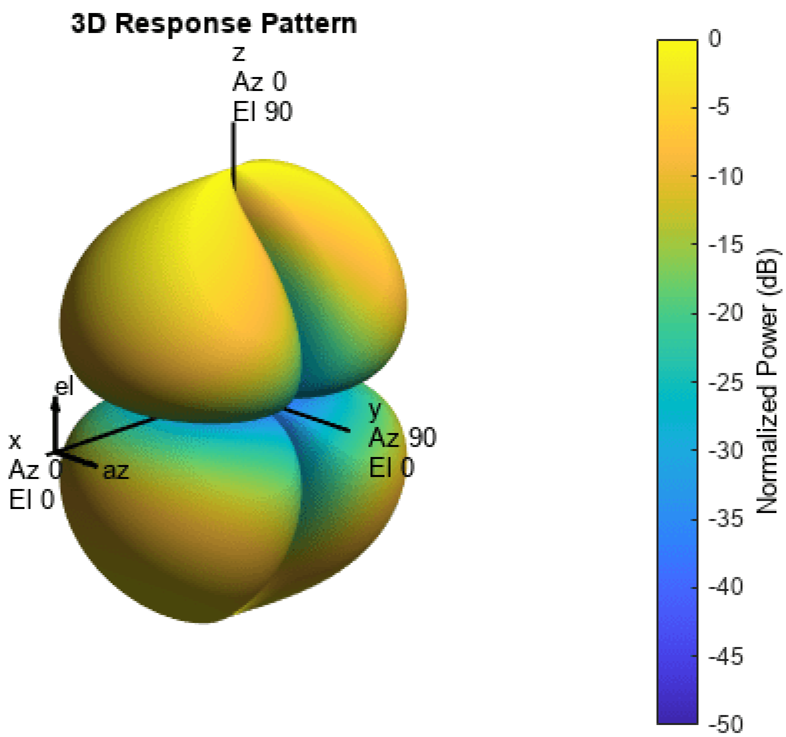
```
    'SpecifyPolarizationPattern',true,...
    'HorizontalMagnitudePattern',horz_pat_azel,...
    'VerticalMagnitudePattern',vert_pat_azel);
```

Assume the antenna operates at 1 GHz.

```
fc = 1e9;
```

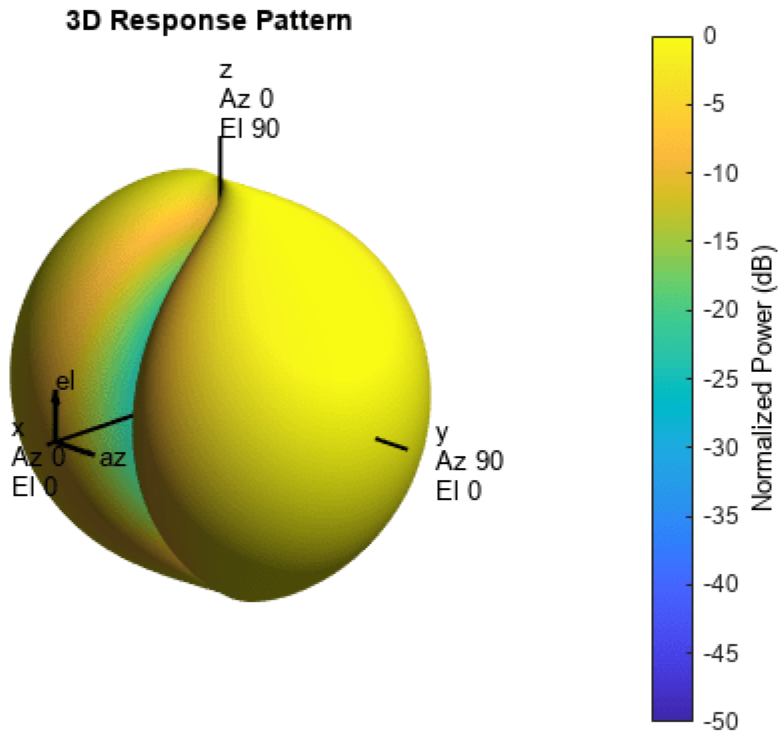
Display the vertical response pattern.

```
pattern(antenna,fc,[-180:180],[-90:90],...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Polarization','V')
```



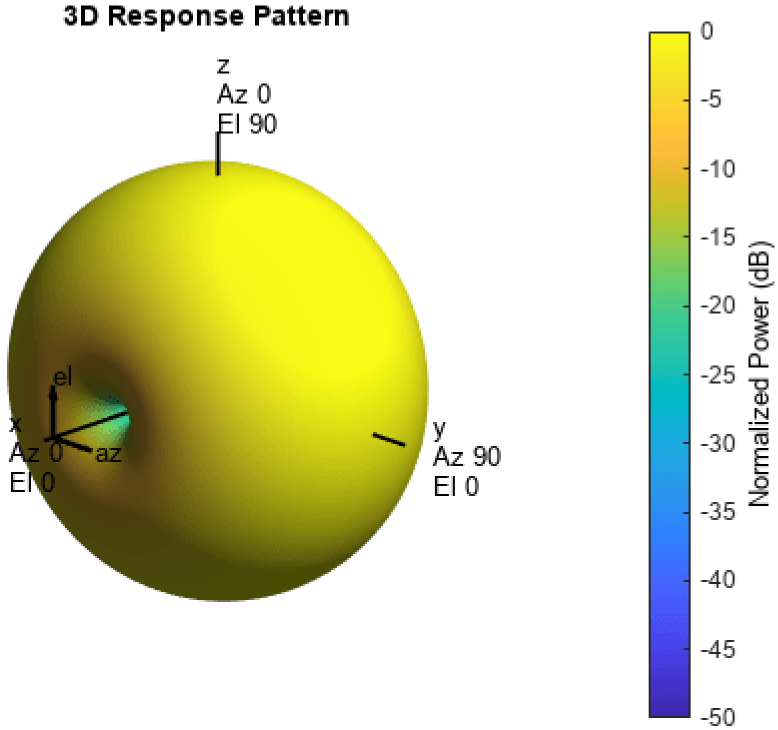
Display the horizontal response pattern.

```
pattern(antenna,fc,[-180:180],[-90:90],...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Polarization','H')
```



The combined polarization response, shown below, illustrates the *x*-axis null of the dipole.

```
pattern(antenna,fc,[-180:180],[-90:90],...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Polarization','combined')
```



Match Custom Antenna Normal to Array Normal

Define a custom antenna in *u*-*v* space. Show how the array response pattern is affected by the choice of the `MatchArrayNormal` property of the `phased.CustomAntennaElement`.

Define the response pattern (in dB) of an antenna as a function of *u* and *v* coordinates within the unit circle. The antenna operates at 1 GHz.

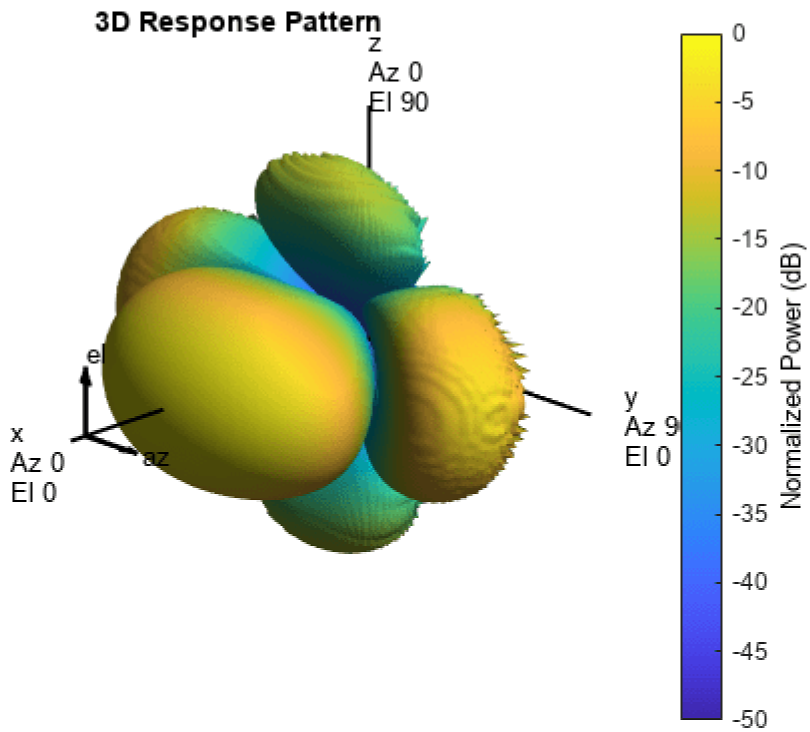
```
fc = 1e9;
c = physconst('LightSpeed');
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Create a custom antenna with this pattern. Convert *u*-*v* coordinates to azimuth and elevation coordinates. Set `MatchArrayNormal` to `false`.

```
[pat_azel,az,el] = uv2azelpat(pat_uv,u,v);
antenna = phased.CustomAntennaElement('AzimuthAngles',az,'ElevationAngles',el, ...
    'MagnitudePattern',mag2db(pat_azel),'PhasePattern',45*ones(size(pat_azel)), ...
    "MatchArrayNormal",false);
```

Construct a 3-by-3 URA with this element and display the antenna pattern in 3-D polar coordinates. The element spacing is one-half wavelength. The array normal points along the *y*-axis.

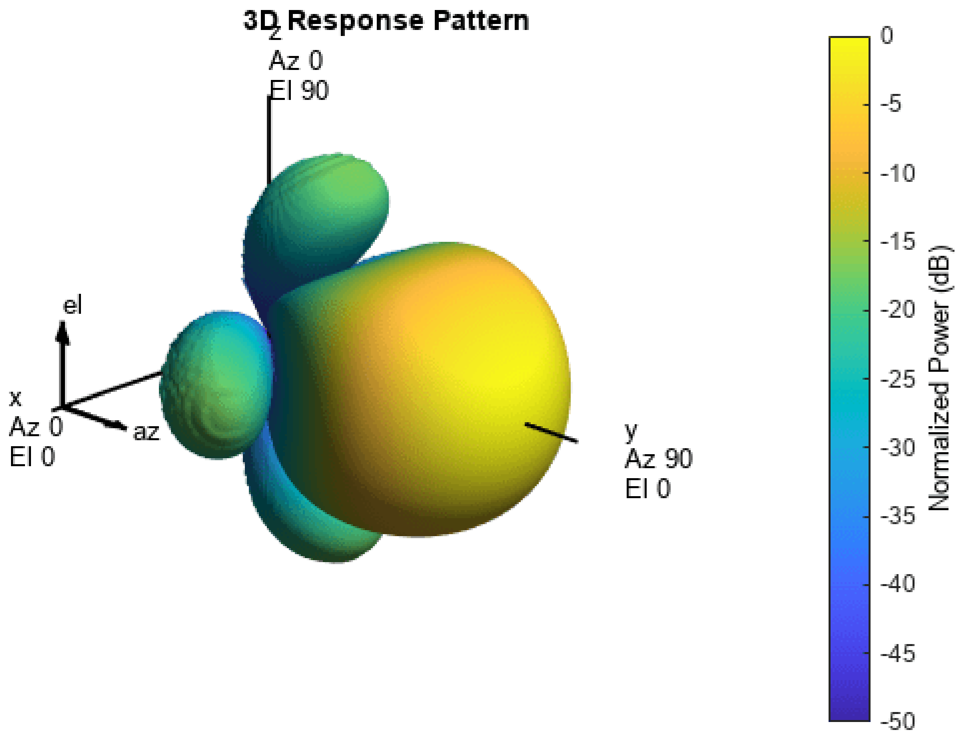
```
lam = c/fc;
array = phased.URA('Element',antenna,'Size',[3 3],'ElementSpacing', ...
    [lam/2 lam/2],'ArrayNormal','y');
pattern(array,fc,-180:180,-90:90,'PropagationSpeed',c, ...
    'CoordinateSystem','polar','Type','powerdb','Normalize',true)
```



The pattern shows the interplay between the element pattern pointing along the *x*-axis and the array pattern pointing along the *y*-axis.

Create another custom antenna with the same radiation pattern. Set `MatchArrayNormal` to true. Then create another array with this element.

```
antenna2 = phased.CustomAntennaElement('AzimuthAngles',az,'ElevationAngles',el, ...
    'MagnitudePattern',mag2db(pat_azel),'PhasePattern',45*ones(size(pat_azel)), ...
    "MatchArrayNormal",true);
array2 = phased.URA('Element',antenna2,'Size',[3 3],'ElementSpacing', ...
    [lam/2 lam/2],'ArrayNormal','y');
pattern(array2,fc,-180:180,-90:90,'PropagationSpeed',c, ...
    'CoordinateSystem','polar','Type','powerdb','Normalize',true)
```



This pattern shows the aligned element and array patterns pointing along the *y*-axis.

Custom Antenna Element Response at 30° Elevation

Construct a user-defined antenna with an omnidirectional response in azimuth and a cosine pattern in elevation. The antenna operates at 1 GHz. Plot the response pattern. Then, find the antenna response at 30°.

```
antenna = phased.CustomAntennaElement;
antenna.AzimuthAngles = -180:180;
antenna.ElevationAngles = -90:90;
antenna.MagnitudePattern = mag2db(repmat(cosd(antenna.ElevationAngles)',...
    1,numel(antenna.AzimuthAngles)));
```

Find the response at 30° elevation for an operating frequency of 1 GHz.

```
fc = 1.0e9;
resp = antenna(fc,[0;30])
```

```
resp = 0.8660
```

Antenna with Custom Radiation Pattern

Create a custom antenna element object. The radiation pattern has a cosine dependence on elevation angle but is independent of azimuth angle.

```
az = -180:90:180;
el = -90:45:90;
elresp = cosd(el);
magpattern = mag2db(repmat(elresp',1,numel(az)));
phasepattern = zeros(size(magpattern));
antenna = phased.CustomAntennaElement('AzimuthAngles',az,...
    'ElevationAngles',el,'MagnitudePattern',magpattern, ...
    'PhasePattern',phasepattern);
```

Display the radiation pattern.

```
disp(antenna.MagnitudePattern)
```

```
      -Inf      -Inf      -Inf      -Inf      -Inf
   -3.0103   -3.0103   -3.0103   -3.0103   -3.0103
         0         0         0         0         0
   -3.0103   -3.0103   -3.0103   -3.0103   -3.0103
      -Inf      -Inf      -Inf      -Inf      -Inf
```

Calculate the antenna response at the azimuth-elevation pairs *(-30,0)* and *(-45,0)* at 500 MHz.

```
ang = [-30 0; -45 0];
resp = antenna(500.0e6,ang);
disp(resp)
```

```
    0.7071
    1.0000
```

The following code illustrates how nearest-neighbor interpolation is used to find the antenna voltage response in the two directions. The total response is the product of the angular response and the frequency response.

```
g = interp2(deg2rad(antenna.AzimuthAngles),...
    deg2rad(antenna.ElevationAngles),...
    db2mag(antenna.MagnitudePattern),...
    deg2rad(ang(1,:))', deg2rad(ang(2,:))','nearest',0);
h = interp1(antenna.FrequencyVector,...
    db2mag(antenna.FrequencyResponse),500e6,'nearest',0);
antresp = h.*g;
```

Compare the value of `antresp` to the response of the antenna.

```
disp(mag2db(antresp))
```

```
   -3.0103
         0
```

Directivity of Custom Antenna

Compute the directivity of a custom antenna element.

Define an antenna pattern for a custom antenna element in azimuth-elevation space. The pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna operates at 1 GHz. Get the response at zero degrees azimuth and from -30 to 30 degrees elevation.

```
fc = 1e9;
azang = [-180:180];
elang = [-90:90];
magpattern = mag2db(repmat(cosd(elang)',1,numel(azang)));
phasepattern = zeros(size(magpattern));
antenna = phased.CustomAntennaElement('AzimuthAngles',azang, ...
    'ElevationAngles',elang,'MagnitudePattern',magpattern, ...
    'PhasePattern',phasepattern);
```

Calculate the directivities as a function of elevation for 0° azimuth angle.

```
angs = [0,0,0,0,0,0,0;-30,-20,-10,0,10,20,30];
freq = 1e9;
d = directivity(antenna,freq,angs)
```

```
d = 7×1

    0.5115
    1.2206
    1.6279
    1.7609
    1.6279
    1.2206
    0.5115
```

The directivity is maximum at 0° elevation.

Custom Antenna Element Supports Polarization

Show that the `CustomAntennaElement` antenna element supports polarization when the `SpecifyPolarizationPattern` property is set to `true`.

```
antenna = phased.CustomAntennaElement('SpecifyPolarizationPattern',true);
isPolarizationCapable(antenna)
```

```
ans = logical
   1
```

The returned value `1` shows that this antenna element supports polarization.

Power and Directivity Patterns of Custom Antenna

Create a custom antenna with a cosine pattern. Show the response at boresight. Then, plot the antenna's field and directivity patterns.

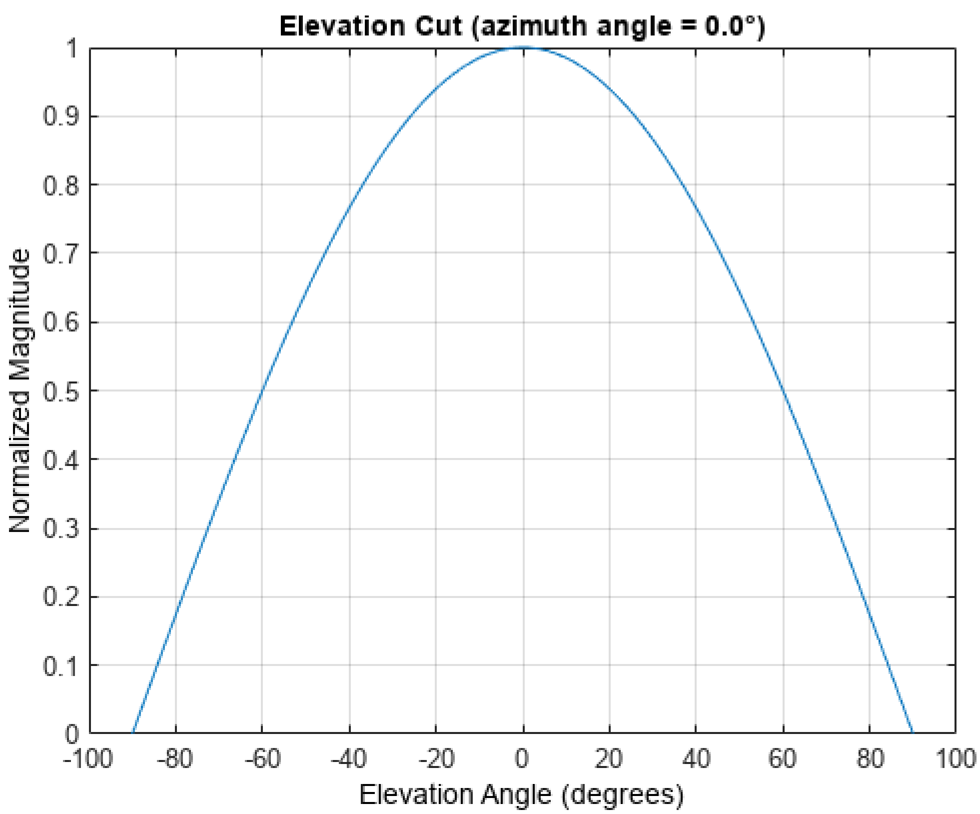
Create the antenna and calculate the response. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna works at 1 GHz.

```
fc = 1e9;
antenna = phased.CustomAntennaElement;
antenna.AzimuthAngles = -180:180;
antenna.ElevationAngles = -90:90;
antenna.MagnitudePattern = mag2db(repmat(cosd(antenna.ElevationAngles)', ...
    1,numel(antenna.AzimuthAngles)));
resp = antenna(fc,[0;0])
```

```
resp = 1
```

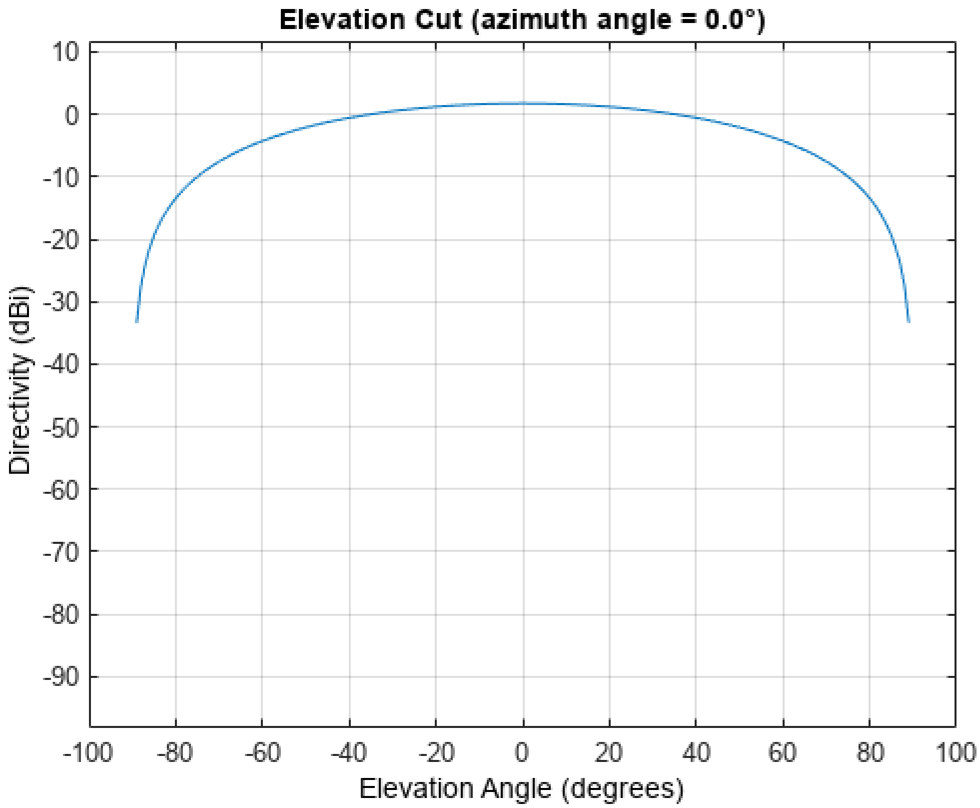
Plot an elevation cut of the magnitude response as a line plot.

```
pattern(antenna,fc,0,[-90:90],'CoordinateSystem','rectangular', ...
    'Type','efield')
```



Plot an elevation cut of the directivity as a line plot, showing that the maximum directivity is approximately 2 dB.

```
pattern(antenna,fc,0,[-90:90],'CoordinateSystem','rectangular', ...
    'Type','directivity')
```



Pattern of Custom Antenna over Selected Range of Angles

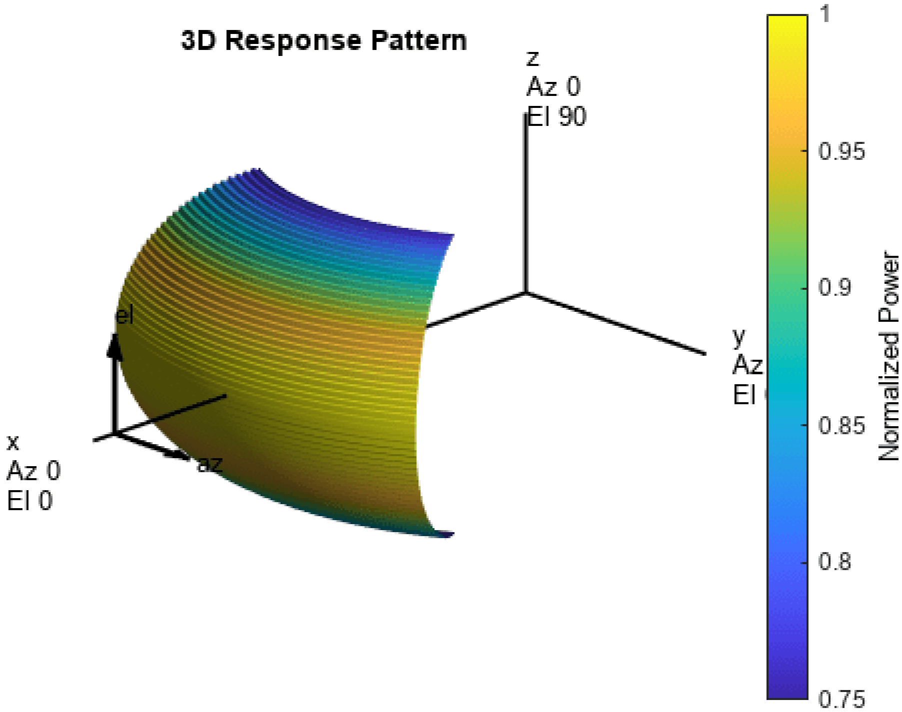
Create a custom antenna System object™. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna operates at a frequency of 1 GHz. First show the response at boresight. Display the 3-D pattern for a 60 degree range of azimuth and elevation angles centered at 0 degrees azimuth and 0 degrees elevation in 0.1 degree increments.

```
fc = 1e9;
azang = -180:180;
elang = -90:90;
magpattern = mag2db(repmat(cosd(elang)',1,numel(azang)));
antenna = phased.CustomAntennaElement('AzimuthAngles',azang, ...
    'ElevationAngles',elang,'MagnitudePattern',magpattern);
resp = antenna(fc,[0;0])
```

```
resp = 1
```

Plot the power pattern for a range of angles.

```
pattern(antenna,fc,[-30:0.1:30],[-30:0.1:30],'CoordinateSystem','polar', ...
    'Type','power')
```

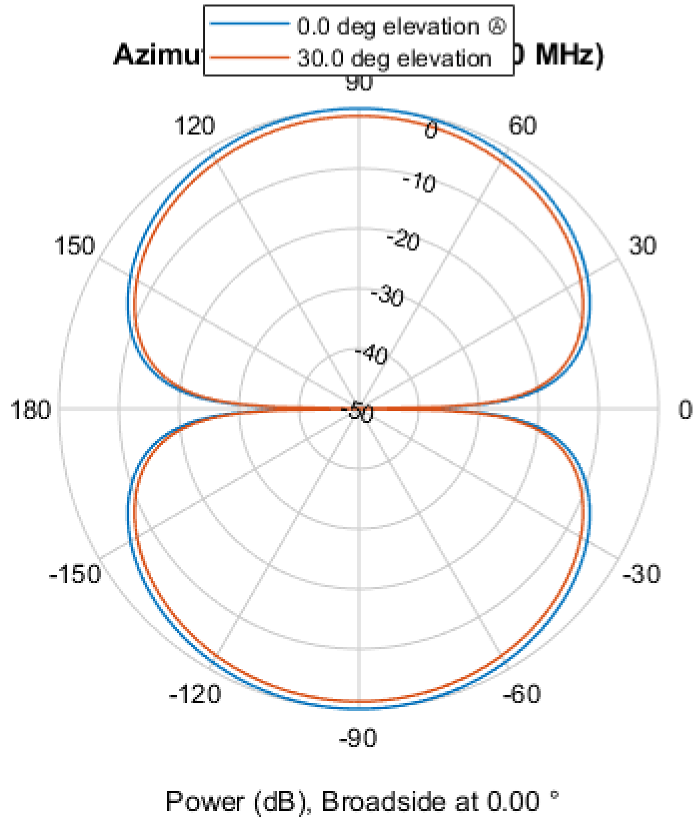


Reduced Azimuth Pattern of Custom Antenna Element

Create an antenna with a custom response. The user-defined pattern has a sine pattern in the azimuth direction and a cosine pattern in the elevation direction. Assume the antenna operates at a frequency of 500 MHz. Plot an azimuth cut of the power pattern of the custom antenna element at 0 and 30 degrees elevation. Assume the operating frequency is 500 MHz.

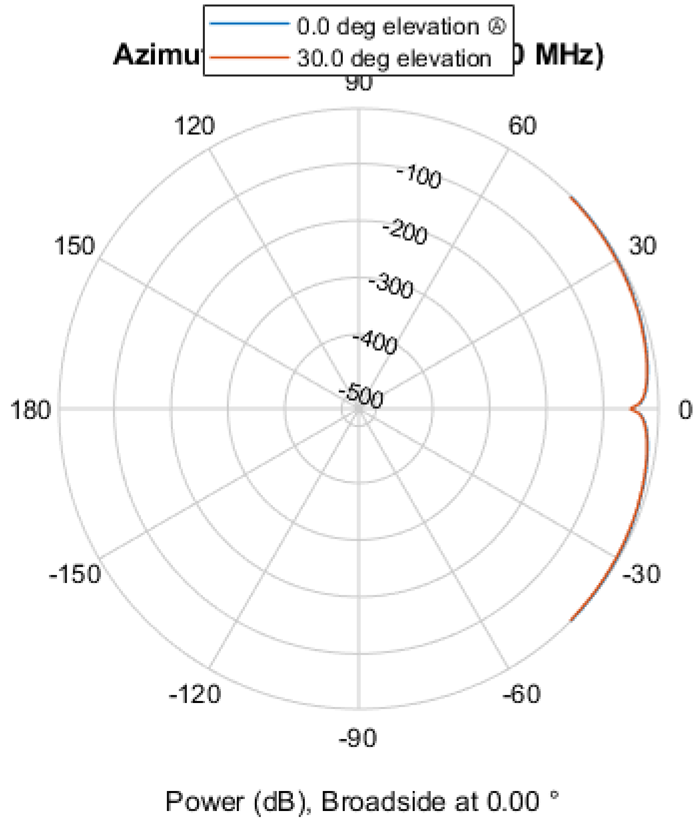
Create the antenna element.

```
fc = 500e6;
antenna = phased.CustomAntennaElement;
antenna.AzimuthAngles = -180:180;
antenna.ElevationAngles = -90:90;
antenna.MagnitudePattern = mag2db(abs(cosd(antenna.ElevationAngles)'*sind(antenna.AzimuthAngles)
patternAzimuth(antenna,fc,[0 30],'Type','powerdb')
```



Plot a reduced range of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(antenna,fc,[0 30],'Azimuth',[-45:45],'Type','powerdb')
```

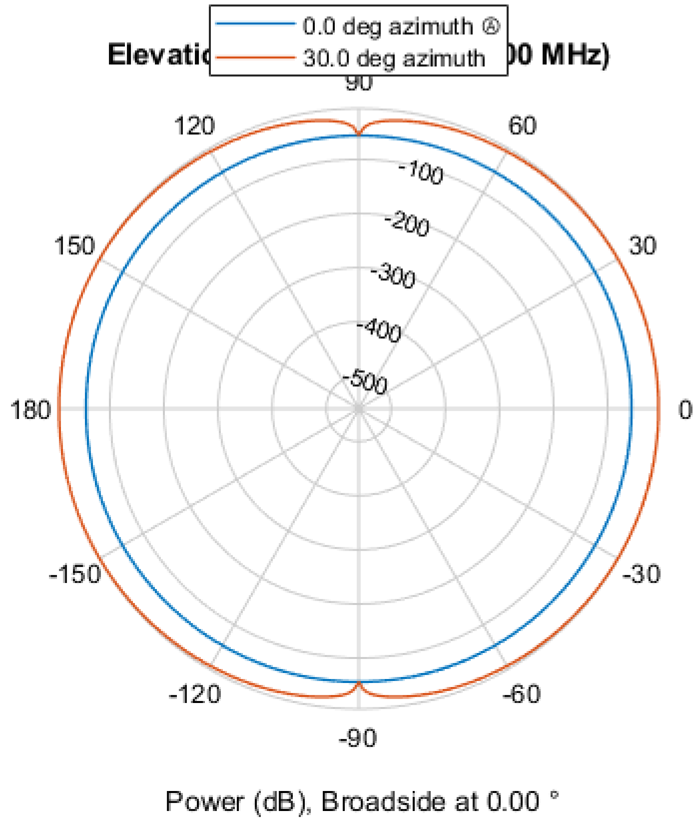


Reduced Elevation Pattern of Custom Antenna Element

Create an antenna with a custom response. The user-defined pattern has a sine pattern in the azimuth direction and a cosine pattern in the elevation direction. Assume the antenna operates at a frequency of 500 MHz. Plot an elevation cut of the power of the custom antenna element at 0 and 30 degrees elevation. Assume the operating frequency is 500 MHz.

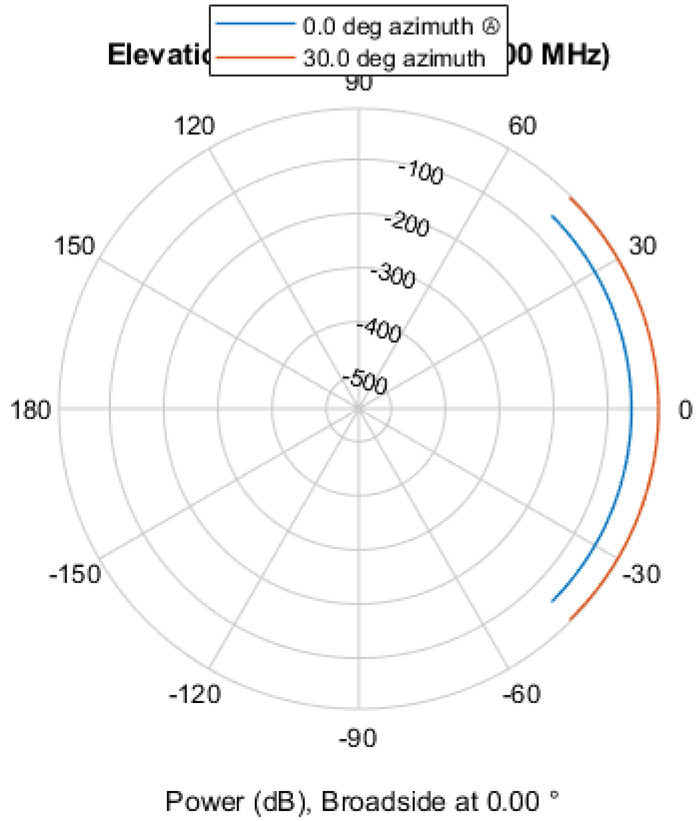
Create the antenna element.

```
fc = 500e6;
antenna = phased.CustomAntennaElement;
antenna.AzimuthAngles = -180:180;
antenna.ElevationAngles = -90:90;
antenna.MagnitudePattern = mag2db(abs(cosd(antenna.ElevationAngles)'*sind(antenna.AzimuthAngles)
patternElevation(antenna,fc,[0 30],'Type','powerdb')
```



Plot a reduced range of elevation angles using the `Azimuth` parameter.

```
patternElevation(antenna,fc,[0 30],'Elevation',[-45:45],'Type','powerdb')
```



Algorithms

The total response of a custom antenna element is a combination of its frequency response and spatial response. `phased.CustomAntennaElement` calculates both responses using nearest neighbor interpolation, and then multiplies the responses to form the total response.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, and `plotResponse` methods are not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

phased.ConformalArray | phased.CrossedDipoleAntennaElement | phased.CosineAntennaElement | phased.IsotropicAntennaElement | phased.ShortDipoleAntennaElement | phased.ULA | phased.URA | uv2azelpat | phitheta2azelpat | uv2azel | phitheta2azel

directivity

System object: phased.CustomAntennaElement
Package: phased

Directivity of custom antenna element

Syntax

```
D = directivity(H,FREQ,ANGLE)
```

Description

D = directivity(H,FREQ,ANGLE) returns the "Directivity" on page 1-295 of a custom antenna element, H, at frequencies specified by FREQ and in direction angles specified by ANGLE.

Input Arguments

H — Custom antenna element

System object

Custom antenna element specified as a phased.CustomAntennaElement System object.

Example: H = phased.CustomAntennaElement;

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.
- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: [1e8 2e6]

Data Types: double

ANGLE — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If ANGLE is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, [az;el]. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: `[45 60; 0 10]`

Data Types: `double`

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Custom Antenna

Compute the directivity of a custom antenna element.

Define an antenna pattern for a custom antenna element in azimuth-elevation space. The pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna operates at 1 GHz. Get the response at zero degrees azimuth and from -30 to 30 degrees elevation.

```
fc = 1e9;
azang = [-180:180];
elang = [-90:90];
magpattern = mag2db(repmat(cosd(elang)',1,numel(azang)));
phasepattern = zeros(size(magpattern));
antenna = phased.CustomAntennaElement('AzimuthAngles',azang, ...
    'ElevationAngles',elang,'MagnitudePattern',magpattern, ...
    'PhasePattern',phasepattern);
```

Calculate the directivities as a function of elevation for 0° azimuth angle.

```
angs = [0,0,0,0,0,0,0;-30,-20,-10,0,10,20,30];
freq = 1e9;
d = directivity(antenna,freq,angs)
```

```
d = 7×1

    0.5115
    1.2206
    1.6279
    1.7609
    1.6279
    1.2206
```

```
0.5115
```

The directivity is maximum at 0˚ elevation.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternElevation` | `patternAzimuth`

isPolarizationCapable

System object: phased.CustomAntennaElement
Package: phased

Polarization capability

Syntax

```
flag = isPolarizationCapable(antenna)
```

Description

`flag = isPolarizationCapable(antenna)` returns a Boolean value, `flag`, indicating whether the `phased.CustomAntennaElement` System object supports polarization. An antenna element supports polarization if it can create or respond to polarized fields. This antenna object supports both polarized and nonpolarized fields.

Input Arguments

antenna — Custom antenna element

`phased.CustomAntennaElement` System object

Custom antenna element, specified as a `phased.CustomAntennaElement` System object.

Output Arguments

flag — Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not. The returned value depends upon the value of the `SpecifyPolarizationPattern` property. If `SpecifyPolarizationPattern` is `true`, then `flag` is `true`. Otherwise it is `false`.

Examples

Custom Antenna Element Supports Polarization

Show that the `CustomAntennaElement` antenna element supports polarization when the `SpecifyPolarizationPattern` property is set to `true`.

```
antenna = phased.CustomAntennaElement('SpecifyPolarizationPattern',true);
isPolarizationCapable(antenna)
```

```
ans = logical
   1
```

The returned value `1` shows that this antenna element supports polarization.

pattern

System object: `phased.CustomAntennaElement`
Package: `phased`

Plot custom antenna element directivity and patterns

Syntax

```
pattern(sElem,FREQ)
pattern(sElem,FREQ,AZ)
pattern(sElem,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sElem,FREQ)` plots the 3-D array directivity pattern (in dBi) for the element specified in `sElem`. The operating frequency is specified in `FREQ`.

`pattern(sElem,FREQ,AZ)` plots the element directivity pattern at the specified azimuth angle.

`pattern(sElem,FREQ,AZ,EL)` plots the element directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the element pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See “Convert plotResponse to pattern” on page 1-305 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sElem` — Custom antenna element
System object

Custom antenna element, specified as a `phased.CustomAntennaElement` System object.

Example: `sElem = phased.CustomAntennaElement;`

`FREQ` — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the

pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type

`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

Normalize — Display normalize pattern

`true` (default) | `false`

Display normalized pattern, specified as the comma-separated pair consisting of `'Normalize`' and a Boolean. Set this parameter to `true` to display a normalized pattern. This parameter does not apply when you set `'Type'` to `'directivity'`. Directivity patterns are already normalized.

Data Types: `logical`

PlotStyle — Plotting style

`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

Polarization — Polarized field component

`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

Output Arguments

PAT — Element pattern
N-by-*M* real-valued matrix

Element pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of PAT correspond to the azimuth angles in the vector specified by EL_ANG. The columns correspond to the elevation angles in the vector specified by AZ_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

Examples

Power and Directivity Patterns of Custom Antenna

Create a custom antenna with a cosine pattern. Show the response at boresight. Then, plot the antenna's field and directivity patterns.

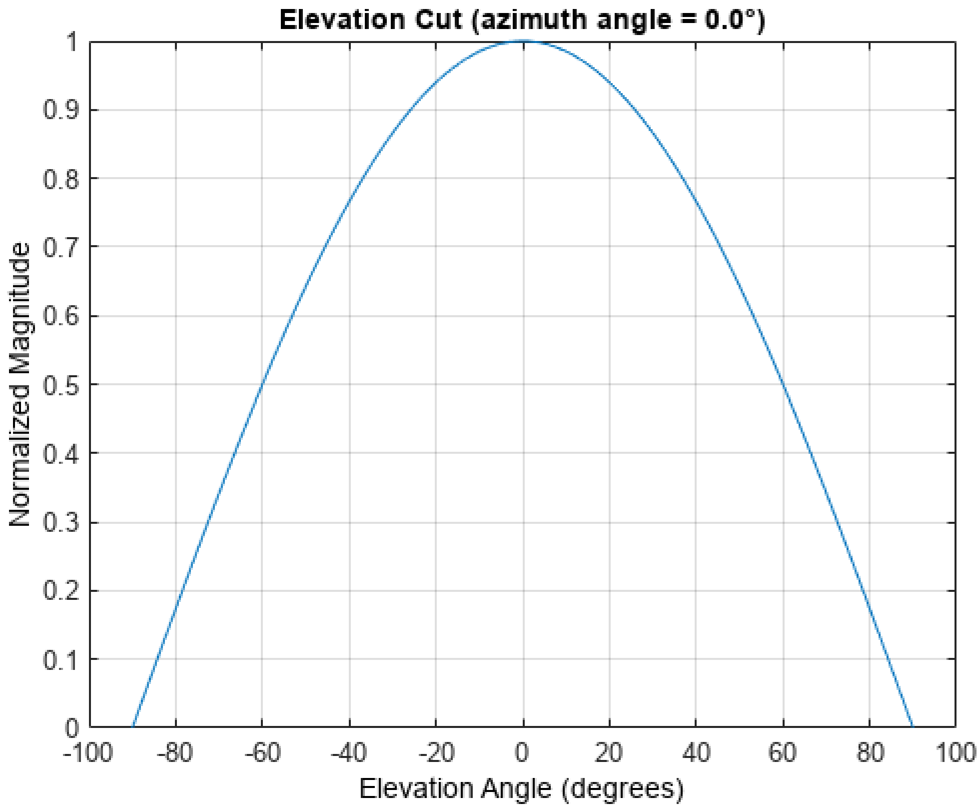
Create the antenna and calculate the response. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna works at 1 GHz.

```
fc = 1e9;
antenna = phased.CustomAntennaElement;
antenna.AzimuthAngles = -180:180;
antenna.ElevationAngles = -90:90;
antenna.MagnitudePattern = mag2db(repmat(cosd(antenna.ElevationAngles)', ...
    1,numel(antenna.AzimuthAngles)));
resp = antenna(fc,[0;0])
```

```
resp = 1
```

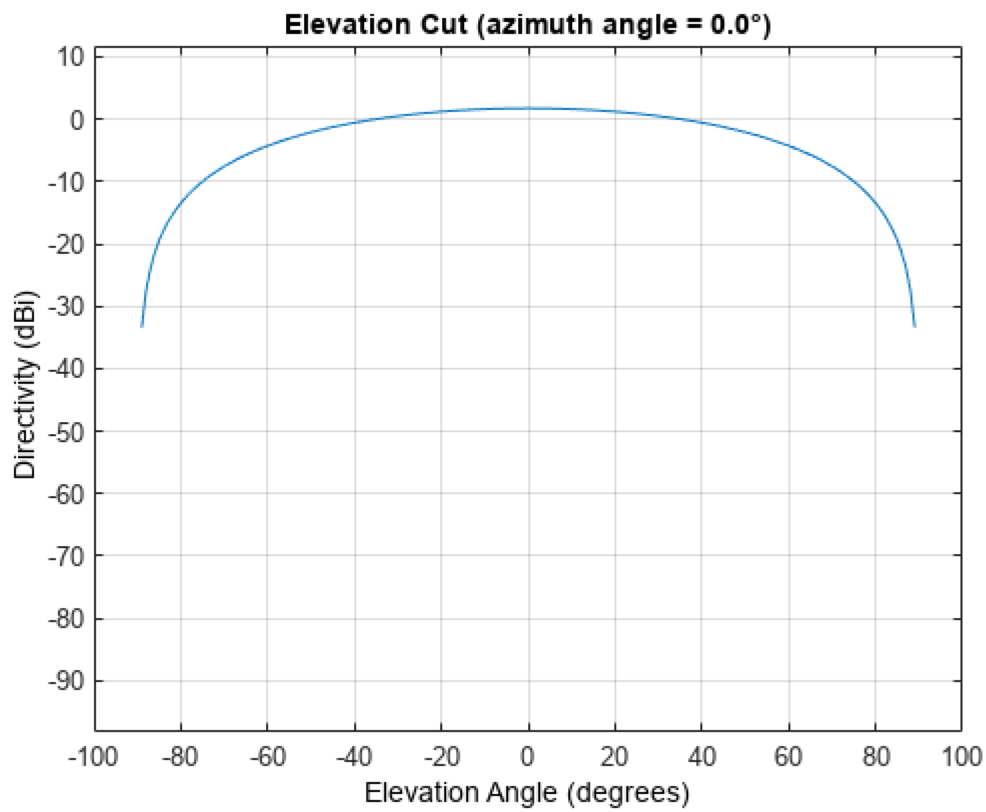
Plot an elevation cut of the magnitude response as a line plot.

```
pattern(antenna,fc,0,[-90:90],'CoordinateSystem','rectangular', ...
    'Type','efield')
```



Plot an elevation cut of the directivity as a line plot, showing that the maximum directivity is approximately 2 dB.

```
pattern(antenna,fc,0,[-90:90],'CoordinateSystem','rectangular', ...
    'Type','directivity')
```



Pattern of Custom Antenna over Selected Range of Angles

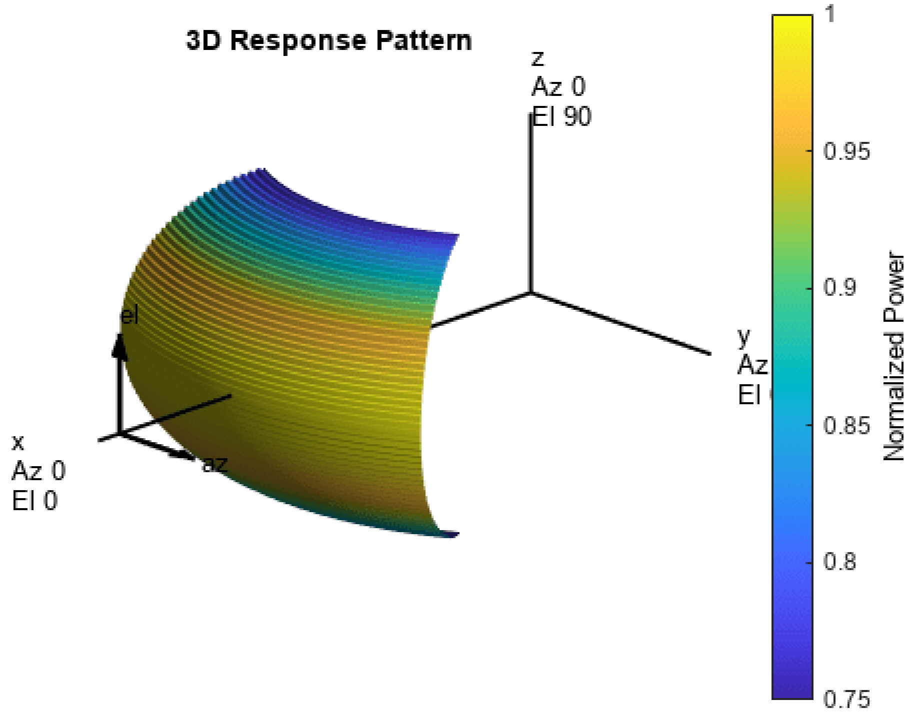
Create a custom antenna System object™. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna operates at a frequency of 1 GHz. First show the response at boresight. Display the 3-D pattern for a 60 degree range of azimuth and elevation angles centered at 0 degrees azimuth and 0 degrees elevation in 0.1 degree increments.

```
fc = 1e9;
azang = -180:180;
elang = -90:90;
magpattern = mag2db(repmat(cosd(elang)',1,numel(azang)));
antenna = phased.CustomAntennaElement('AzimuthAngles',azang, ...
    'ElevationAngles',elang,'MagnitudePattern',magpattern);
resp = antenna(fc,[0;0])
```

```
resp = 1
```

Plot the power pattern for a range of angles.

```
pattern(antenna,fc,[-30:0.1:30],[-30:0.1:30],'CoordinateSystem','polar', ...
    'Type','power')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.

Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.

Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.

'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the UV grid values using AZ and EL.</td>
</tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
	Display space 'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.	
'CutAngle' name-value pair	Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.	No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.
'NormalizeResponse' name-value pair	Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.	Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.
'OverlayFreq' name-value pair	Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.	'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot. The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.
'Polarization' name-value pair	Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.	'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.
'Unit' name-value pair	Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.	'Type' name-value pair, uses equivalent options with different names **plotResponse** — **pattern** 'db' — 'powerdb' 'mag' — 'efield' 'pow' — 'power' 'dbi' — 'directivity'
'Weights' name-value pair	Array element tapers (or weights).	'Weights' name-value pair (no change).
'AzimuthAngles' name-value pair	Azimuth angles used to display the antenna or array response.	AZ argument

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternElevation` | `patternAzimuth`

patternAzimuth

System object: `phased.CustomAntennaElement`
Package: `phased`

Plot custom antenna element directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sElem,FREQ)
patternAzimuth(sElem,FREQ,EL)
patternAzimuth(sElem,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sElem,FREQ)` plots the 2-D element directivity pattern versus azimuth (in dBi) for the element `sElem` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

`patternAzimuth(sElem,FREQ,EL)`, in addition, plots the 2-D element directivity pattern versus azimuth (in dBi) at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sElem,FREQ,EL,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sElem` — Custom antenna element
System object

Custom antenna element, specified as a `phased.CustomAntennaElement` System object.

Example: `sElem = phased.CustomAntennaElement;`

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

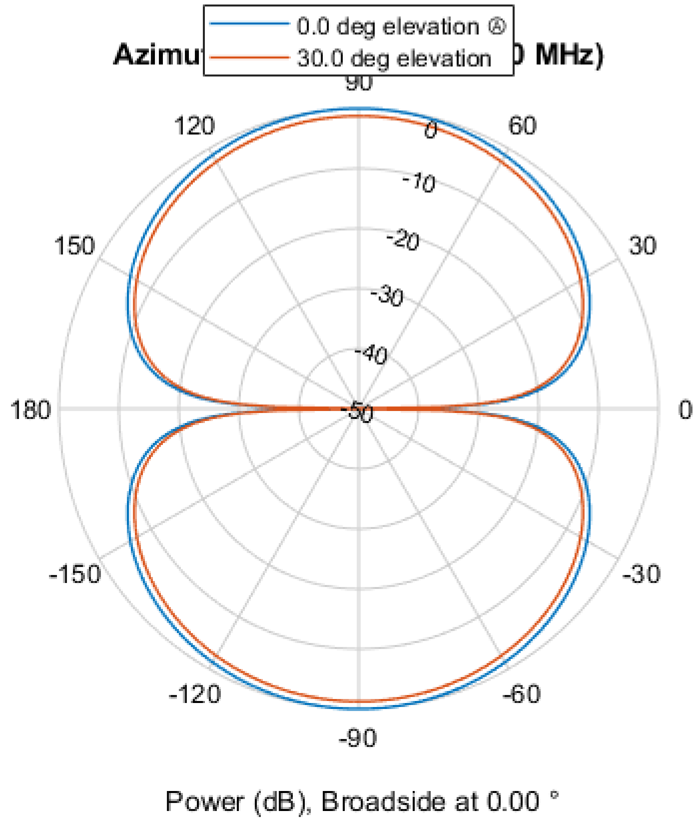
Examples

Reduced Azimuth Pattern of Custom Antenna Element

Create an antenna with a custom response. The user-defined pattern has a sine pattern in the azimuth direction and a cosine pattern in the elevation direction. Assume the antenna operates at a frequency of 500 MHz. Plot an azimuth cut of the power pattern of the custom antenna element at 0 and 30 degrees elevation. Assume the operating frequency is 500 MHz.

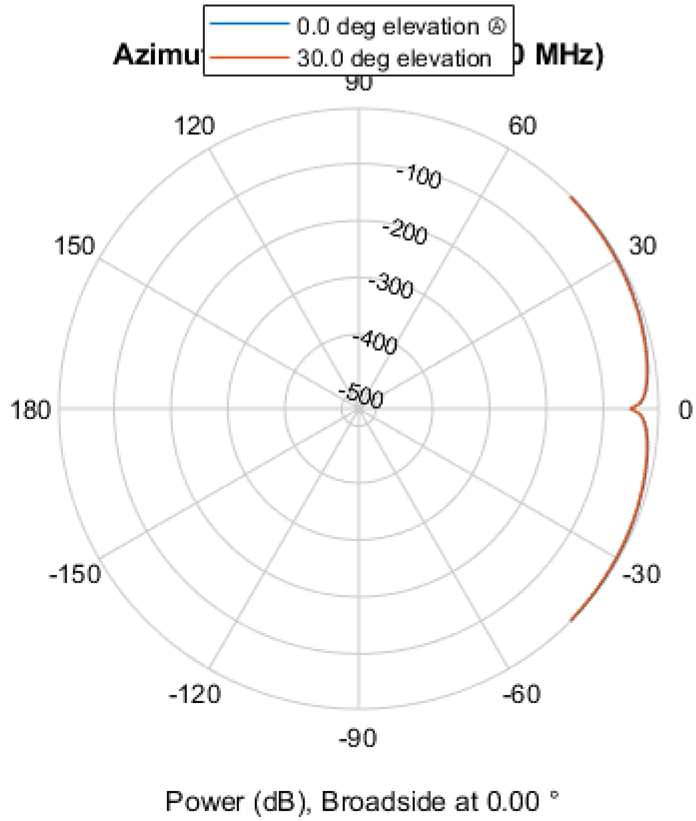
Create the antenna element.

```
fc = 500e6;
antenna = phased.CustomAntennaElement;
antenna.AzimuthAngles = -180:180;
antenna.ElevationAngles = -90:90;
antenna.MagnitudePattern = mag2db(abs(cosd(antenna.ElevationAngles)'*sind(antenna.AzimuthAngles)
patternAzimuth(antenna,fc,[0 30],'Type','powerdb')
```



Plot a reduced range of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(antenna,fc,[0 30],'Azimuth',[-45:45],'Type','powerdb')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: `phased.CustomAntennaElement`
Package: `phased`

Plot custom antenna element directivity or pattern versus elevation

Syntax

```
patternElevation(sElem,FREQ)
patternElevation(sElem,FREQ,AZ)
patternElevation(sElem,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sElem,FREQ)` plots the 2-D element directivity pattern versus elevation (in dBi) for the element `sElem` at zero degrees azimuth angle. The argument `FREQ` specifies the operating frequency.

`patternElevation(sElem,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sElem,FREQ,AZ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sElem` — Custom antenna element

System object

Custom antenna element, specified as a `phased.CustomAntennaElement` System object.

Example: `sElem = phased.CustomAntennaElement;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

`AZ` — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Type` — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

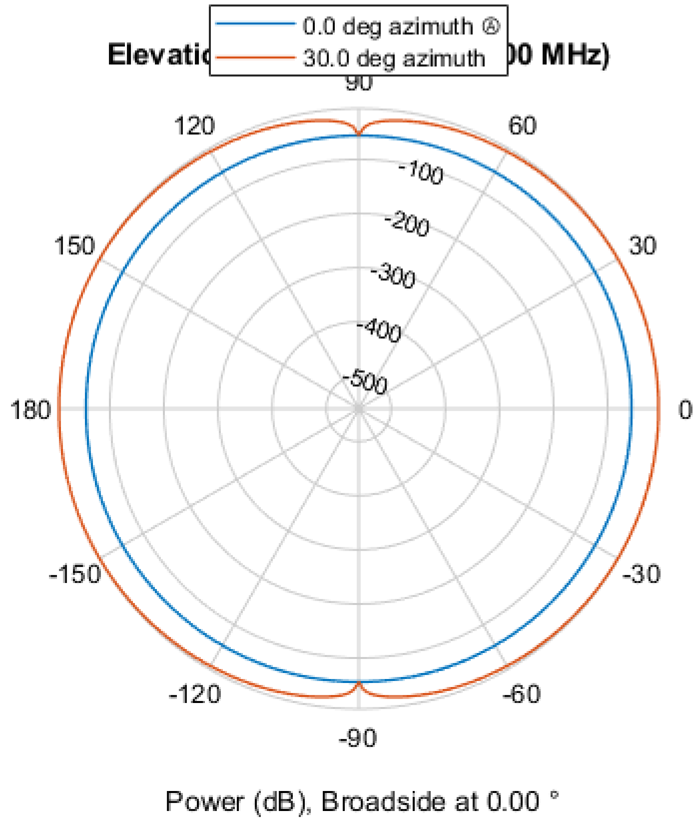
Examples

Reduced Elevation Pattern of Custom Antenna Element

Create an antenna with a custom response. The user-defined pattern has a sine pattern in the azimuth direction and a cosine pattern in the elevation direction. Assume the antenna operates at a frequency of 500 MHz. Plot an elevation cut of the power of the custom antenna element at 0 and 30 degrees elevation. Assume the operating frequency is 500 MHz.

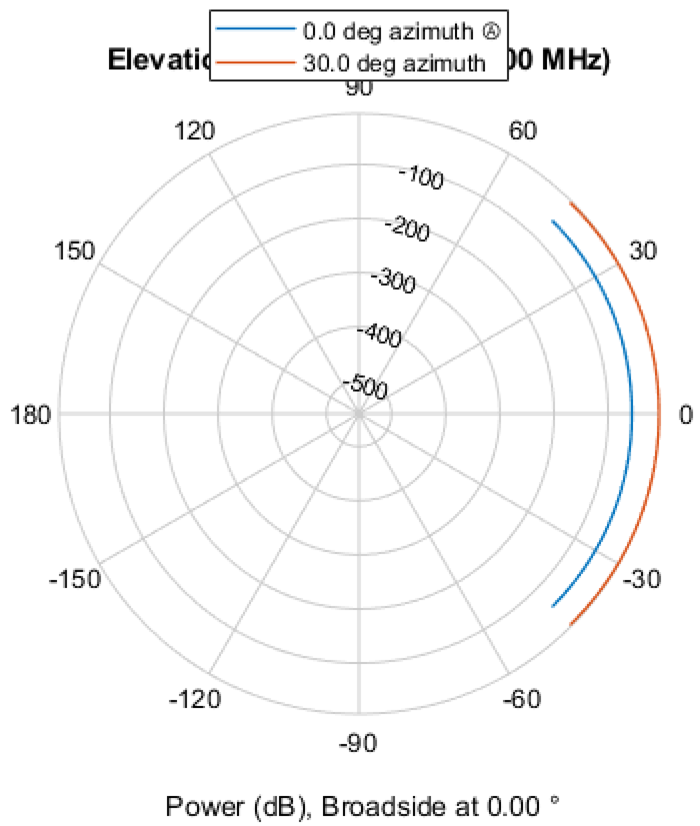
Create the antenna element.

```
fc = 500e6;
antenna = phased.CustomAntennaElement;
antenna.AzimuthAngles = -180:180;
antenna.ElevationAngles = -90:90;
antenna.MagnitudePattern = mag2db(abs(cosd(antenna.ElevationAngles)'*sind(antenna.AzimuthAngles))
patternElevation(antenna,fc,[0 30],'Type','powerdb')
```



Plot a reduced range of elevation angles using the `Azimuth` parameter.

```
patternElevation(antenna,fc,[0 30],'Elevation',[-45:45],'Type','powerdb')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also
`pattern` | `patternAzimuth`

plotResponse

System object: phased.CustomAntennaElement
Package: phased

Plot response pattern of antenna

Syntax

```
plotResponse(H,FREQ)
plotResponse(H,FREQ,Name,Value)
hPlot = plotResponse( ___ )
```

Description

plotResponse(H,FREQ) plots the element response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in FREQ.

plotResponse(H,FREQ,Name,Value) plots the element response with additional options specified by one or more Name,Value pair arguments.

hPlot = plotResponse(___) returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Element System object

FREQ

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. FREQ must lie within the range specified by the FrequencyVector property of H. If you set the 'RespCut' property of H to '3D', FREQ must be a scalar. When FREQ is a row vector, plotResponse draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

CutAngle

Cut angle specified as a scalar. This argument is applicable only when RespCut is 'Az' or 'El'. If RespCut is 'Az', CutAngle must be between –90 and 90. If RespCut is 'El', CutAngle must be between –180 and 180.

Default: 0

`Format`

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

`NormalizeResponse`

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

`OverlayFreq`

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

`Polarization`

Specify the polarization options for plotting the antenna response pattern. The allowable values are `|'None' | 'Combined' | 'H' | 'V' |` where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

`RespCut`

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

`Unit`

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

`AzimuthAngles`

Azimuth angles for plotting element response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

`ElevationAngles`

Elevation angles for plotting element response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

`UGrid`

U coordinate values for plotting element response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

`VGrid`

V coordinate values for plotting element response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Plot Response and Directivity of Custom Antenna

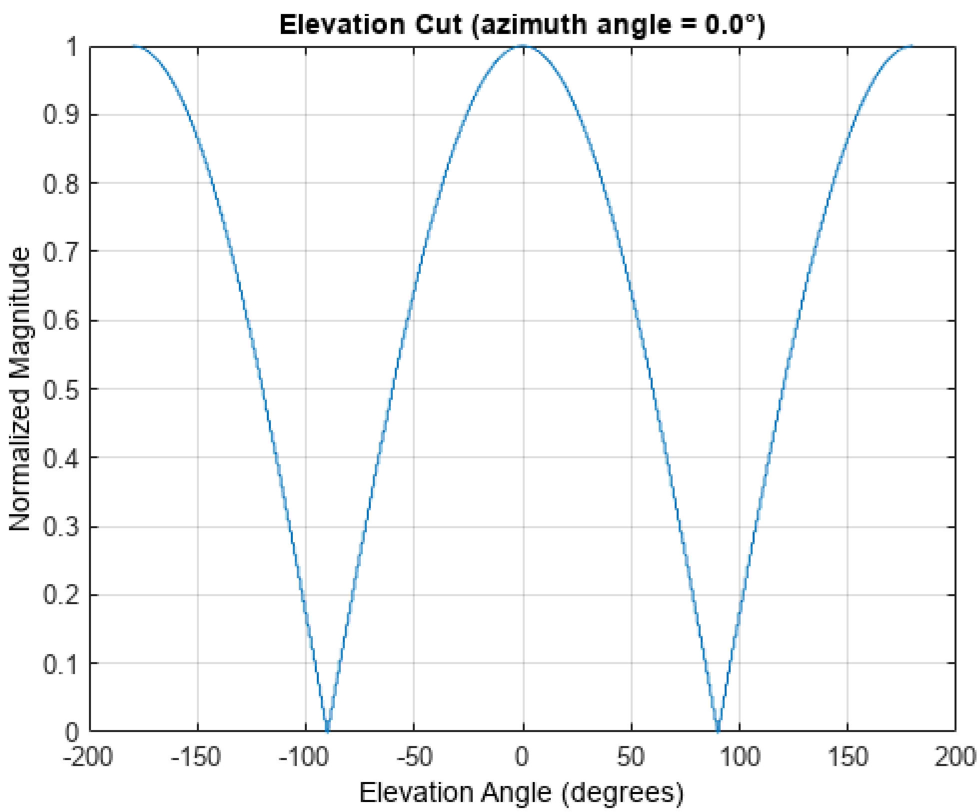
Create a custom antenna with a cosine pattern. Then, plot the antenna's response.

Create the antenna and calculate the response. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna works at 1 GHz.

```
fc = 1e9;
azang = [-180:180];
elang = [-90:90];
magpattern = mag2db(repmat(cosd(elang)',1,numel(azang)));
phasepattern = zeros(size(magpattern));
antenna = phased.CustomAntennaElement('AzimuthAngles',azang, ...
    'ElevationAngles',elang,'MagnitudePattern',magpattern, ...
    'PhasePattern',phasepattern);
```

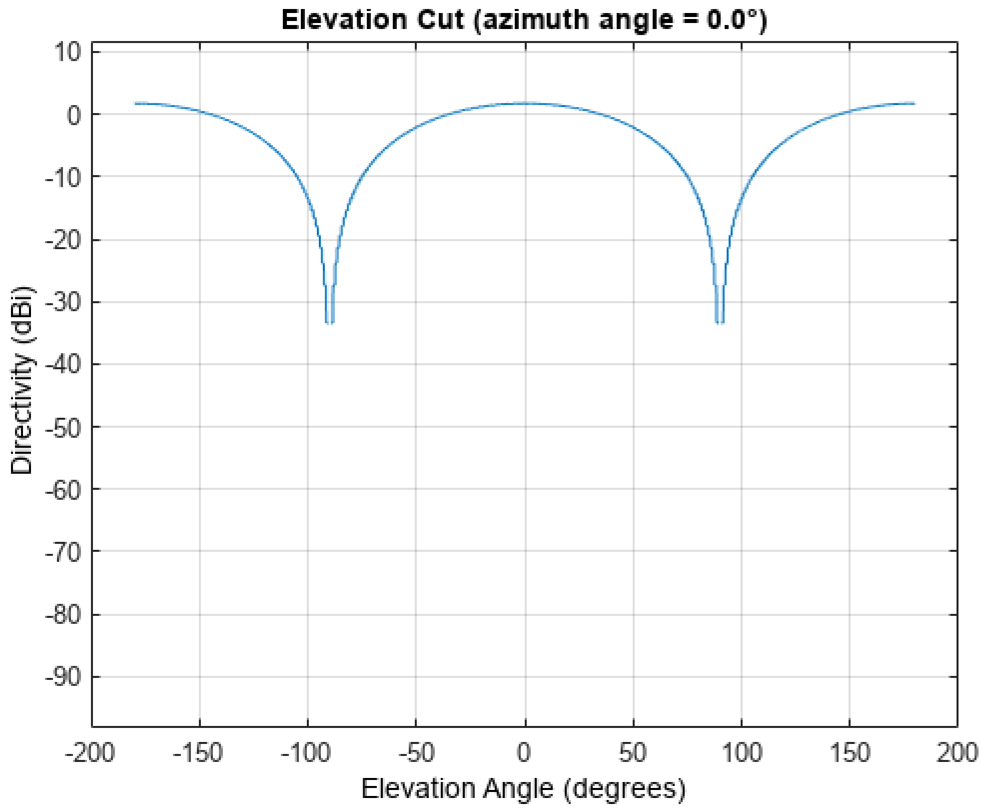
Plot an elevation cut of the magnitude response as a line plot.

```
plotResponse(antenna,fc,'RespCut','El','ElevationAngles',[-90:0.1:90],...
    'Format','Line','Unit','mag')
```



Plot an elevation cut of the directivity as a line plot, showing that the maximum directivity is approximately 2 dB.

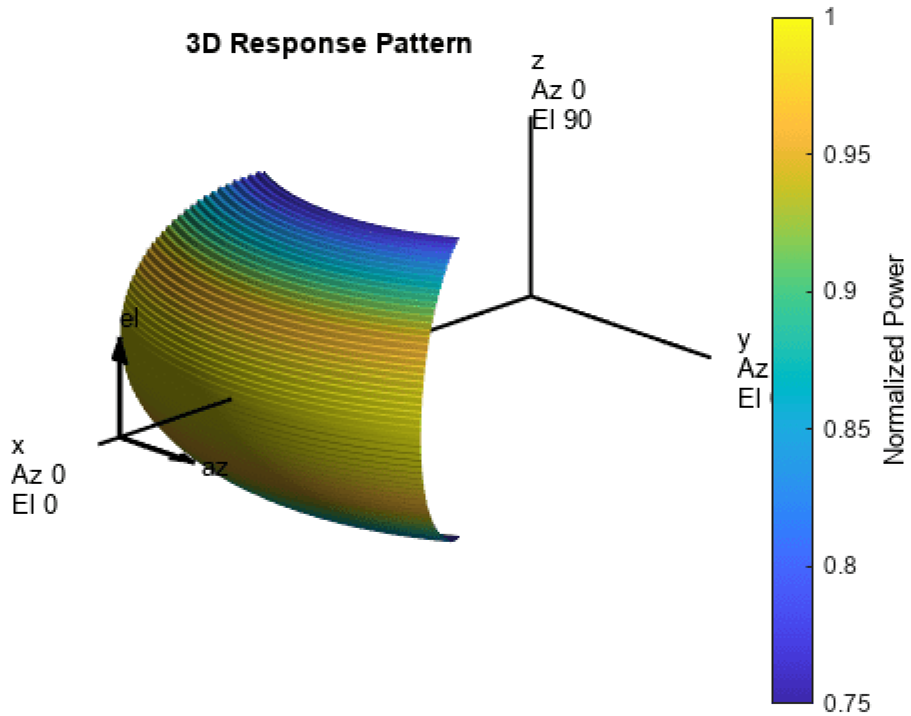
```
plotResponse(antenna,fc,'RespCut','El','ElevationAngles',[-90:0.1:90],...
    'Format','Line','Unit','dbi')
```



Plot Response of Custom Antenna Over Selected Range of Angles

Create an antenna with a custom response. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna operates at a frequency of 1 GHz. Display the 3-D response for a 60 degree range of azimuth and elevation angles centered at 0 degrees azimuth and 0 degrees elevation in 0.1 degree increments.

```
fc = 1e9;
azang = [-180:180];
elang = [-90:90];
magpattern = mag2db(repmat(cosd(elang)',1,numel(azang)));
phasepattern = zeros(size(magpattern));
antenna = phased.CustomAntennaElement('AzimuthAngles',azang, ...
    'ElevationAngles',elang,'MagnitudePattern',magpattern, ...
    'PhasePattern',phasepattern);
resp = antenna(fc,[0;0]);
plotResponse(antenna,fc,'RespCut','3D','AzimuthAngles',[-30:0.1:30],...
    'ElevationAngles',[-30:0.1:30],'Format','Polar','Unit','pow')
```



See Also

uv2azel | azel2uv

step

System object: `phased.CustomAntennaElement`
Package: `phased`

Output response of antenna element

Syntax

```
RESP = step(H,FREQ,ANG)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the antenna's voltage response `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`. The form of `RESP` depends upon whether the antenna element supports polarization as determined by the `SpecifyPolarizationPattern` property. If `SpecifyPolarizationPattern` is set to `false`, `RESP` is an *M*-by-*L* matrix containing the antenna response at the *M* angles specified in `ANG` and at the *L* frequencies specified in `FREQ`. If `SpecifyPolarizationPattern` is set to `true`, `RESP` is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`, representing the antenna's response in horizontal and vertical polarization, respectively. Each field is an *M*-by-*L* matrix containing the antenna response at the *M* angles specified in `ANG` and at the *L* frequencies specified in `FREQ`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Antenna element object.

FREQ

Operating frequencies of antenna in hertz. `FREQ` is a row vector of length L.

ANG

Directions in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If ANG is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If ANG is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

Output Arguments

RESP

Voltage response of antenna element. The output depends on whether the antenna element supports polarization or not.

- If the antenna element does not support polarization, RESP is an *M*-by-*L* matrix. In this matrix, *M* represents the number of angles specified in ANG while *L* represents the number of frequencies specified in FREQ.
- If the antenna element supports polarization, RESP is a MATLAB struct with fields RESP.H and RESP.V containing responses for the horizontal and vertical polarization components of the antenna radiation pattern. RESP.H and RESP.V are *M*-by-*L* matrices. In these matrices, *M* represents the number of angles specified in ANG while *L* represents the number of frequencies specified in FREQ.

Examples

Custom Antenna Element Response at 30° Elevation

Construct a user-defined antenna with an omnidirectional response in azimuth and a cosine pattern in elevation. The antenna operates at 1 GHz. Plot the response pattern. Then, find the antenna response at 30°.

```
antenna = phased.CustomAntennaElement;
antenna.AzimuthAngles = -180:180;
antenna.ElevationAngles = -90:90;
antenna.MagnitudePattern = mag2db(repmat(cosd(antenna.ElevationAngles)',...
    1,numel(antenna.AzimuthAngles)));
```

Find the response at 30° elevation for an operating frequency of 1 GHz.

```
fc = 1.0e9;
resp = antenna(fc,[0;30])
```

```
resp = 0.8660
```

Antenna with Custom Radiation Pattern

Create a custom antenna element object. The radiation pattern has a cosine dependence on elevation angle but is independent of azimuth angle.

```
az = -180:90:180;
el = -90:45:90;
```

```
elresp = cosd(el);
magpattern = mag2db(repmat(elresp',1,numel(az)));
phasepattern = zeros(size(magpattern));
antenna = phased.CustomAntennaElement('AzimuthAngles',az,...
    'ElevationAngles',el,'MagnitudePattern',magpattern, ...
    'PhasePattern',phasepattern);
```

Display the radiation pattern.

```
disp(antenna.MagnitudePattern)
```

```
     -Inf      -Inf      -Inf      -Inf      -Inf
  -3.0103   -3.0103   -3.0103   -3.0103   -3.0103
        0         0         0         0         0
  -3.0103   -3.0103   -3.0103   -3.0103   -3.0103
     -Inf      -Inf      -Inf      -Inf      -Inf
```

Calculate the antenna response at the azimuth-elevation pairs *(-30,0)* and *(-45,0)* at 500 MHz.

```
ang = [-30 0; -45 0];
resp = antenna(500.0e6,ang);
disp(resp)
```

```
    0.7071
    1.0000
```

The following code illustrates how nearest-neighbor interpolation is used to find the antenna voltage response in the two directions. The total response is the product of the angular response and the frequency response.

```
g = interp2(deg2rad(antenna.AzimuthAngles),...
    deg2rad(antenna.ElevationAngles),...
    db2mag(antenna.MagnitudePattern),...
    deg2rad(ang(1,:))', deg2rad(ang(2,:))','nearest',0);
h = interp1(antenna.FrequencyVector,...
    db2mag(antenna.FrequencyResponse),500e6,'nearest',0);
antresp = h.*g;
```

Compare the value of `antresp` to the response of the antenna.

```
disp(mag2db(antresp))
```

```
  -3.0103
        0
```

Algorithms

The total response of a custom antenna element is a combination of its frequency response and spatial response. `phased.CustomAntennaElement` calculates both responses using nearest neighbor interpolation, and then multiplies the responses to form the total response.

See Also

`uv2azel` | `phitheta2azel`

phased.CustomMicrophoneElement

Package: `phased`

Custom microphone element

Description

The `CustomMicrophoneElement` System object models a microphone element with a custom spatial response pattern.

To compute the response of the microphone element for specified directions:

1. Create the `phased.CustomMicrophoneElement` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
microphone = phased.CustomMicrophoneElement
microphone = phased.CustomMicrophoneElement(Name=Value)
```

Description

`microphone = phased.CustomMicrophoneElement` creates a custom microphone System object, `microphone`, with default object properties.

`microphone = phased.CustomMicrophoneElement(Name=Value)` creates a custom microphone object, `microphone`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1=Value1`,...,`NameN=ValueN`).

Example: `microphone = phased.CustomMicrophoneElement(FrequencyVector=[0 1000],FrequencyResponse=[0 -10],PolarPatternFrequencies=[100 1000])` creates a custom microphone element with its frequency response specified at 0 and 1000 Hz. The frequency response at these frequencies is 0 and -10 dB. The pattern frequencies are 100 and 1000 Hz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`FrequencyVector` — Operating frequencies
`[0 1e20]` (default) | 1-by-*L* real-valued vector of positive values

The frequencies where the element responses are measured, specified as 1-by-*L* real-valued vector of positive values. The elements of the vector must be in increasing magnitude. The microphone element has no response outside the specified frequency range. Units are Hz.

Data Types: `double`

`FrequencyResponse` — Frequency responses
`[0 0]` (default) | 1-by-*L* real-valued vector

Frequency responses measured at the frequencies defined in the `FrequencyVector` property, specified as a1-by-*L* real-valued vector. The length of the vector must equal the length of the frequency vector specified in the `FrequencyVector` property. Units are dB.

Data Types: `double`

`PolarPatternFrequencies` — Polar pattern measuring frequencies
`1e3` (default) | 1-by-*M* real-valued vector of positive values

Measuring frequencies of the polar patterns, specified as a1-by-*M* real-valued vector of positive values. The measuring frequencies must be within the frequency range specified in the `FrequencyVector` property. Units are Hz.

Data Types: `double`

`PolarPatternAngles` — Polar pattern measuring angles
`[-180:180]` (default) | 1-by-*N* real-valued vector

Measuring angles in degrees of the polar patterns, specified as a 1-by-*N* real-valued vector. The angles are measured from the central pickup axis of the microphone, and must be between *–180°* and *180°*, inclusive.

Data Types: `double`

`PolarPattern` — Polar pattern
Omnidirectional pattern with 0 dB response (default) | *M*-by-*N* real-valued matrix

Polar patterns of the microphone element, specified as an *M*-by-*N* real-valued matrix. *M* is the number of measuring frequencies specified in the `PolarPatternFrequencies` property. *N* is the number of measuring angles specified in the `PolarPatternAngles` property. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in the `PolarPatternFrequencies` property and corresponding angles specified in the `PolarPatternAngles` property.

The pattern is assumed to be measured in the azimuth plane where the elevation angle is 0 and where the central pickup axis is assumed to be zero degrees azimuth and zero degrees elevation. The polar pattern is assumed to be symmetric around the central axis and therefore the microphone’s response pattern in 3-D space can be constructed from the polar pattern. Units are dB.

Data Types: `double`

Usage

Syntax

`RESP = microphone(FREQ,ANG)`

Description

`RESP = microphone(FREQ,ANG)` returns the microphone's magnitude response, `RESP`, at frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`FREQ` — Signal frequencies
1-by-*P* row vector of positive values

Signal frequencies, specified as a 1-by-*P* row vector of positive values. Units are Hz.

`ANG` — Response directions
1-by-*Q* vector of real-values | 2-by-*Q* matrix of real-values

Response directions, specified as a 1-by-*Q* vector of real-values or a 2-by-*Q* matrix of real-values.

- If `ANG` is a 1-by-*Q* vector, each element specifies a direction's azimuth angle. The azimuth angle must be between *–180°* and *180°*, inclusive. In this case, the corresponding elevation angle is assumed to be zero.
- If `ANG` is a 2-by-Q matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`. The azimuth angle must be between *–180°* and *180°*, inclusive. The elevation angle must lie between *–90°* and *90°*, inclusive.

Units are in degrees.

Output Arguments

`RESP` — Microphone response
Q-by-*P* real-valued matrix

Microphone magnitude response, returned as an *Q*-by-*P* real-valued matrix. The matrix contains the responses of the microphone element at the *Q* angles specified in `ANG` and the *P* frequencies specified in `FREQ`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Custom Microphone Response

Construct a custom cardioid microphone with an operating frequency of 700 Hz. Find the microphone response in the directions: *(0,0)* degrees azimuth and elevation and *(40,50)* degrees azimuth and elevation.

```
microphone = phased.CustomMicrophoneElement;
microphone.PolarPatternFrequencies = [500 1000];
microphone.PolarPattern = mag2db([ ...
    0.5+0.5*cosd(microphone.PolarPatternAngles); ...
    0.6+0.4*cosd(microphone.PolarPatternAngles)]);
fc = 700;
ang = [0 0; 40 50]';
resp = microphone(fc,ang)
```

```
resp = 2×1

    1.0000
    0.7424
```

Custom Cardioid Microphone Response

Create a custom cardioid microphone, and calculate the microphone response at 500, 1500, and 2000 Hz in two directions: *(0,0)* azimuth and elevation, and *(40,50)* azimuth and elevation. Then display the microphone pattern.

```
microphone = phased.CustomMicrophoneElement( ...
    PolarPatternFrequencies=[500 1000]);
microphone.PolarPattern = mag2db([...
    0.5+0.5*cosd(microphone.PolarPatternAngles);...
    0.6+0.4*cosd(microphone.PolarPatternAngles)]);
```

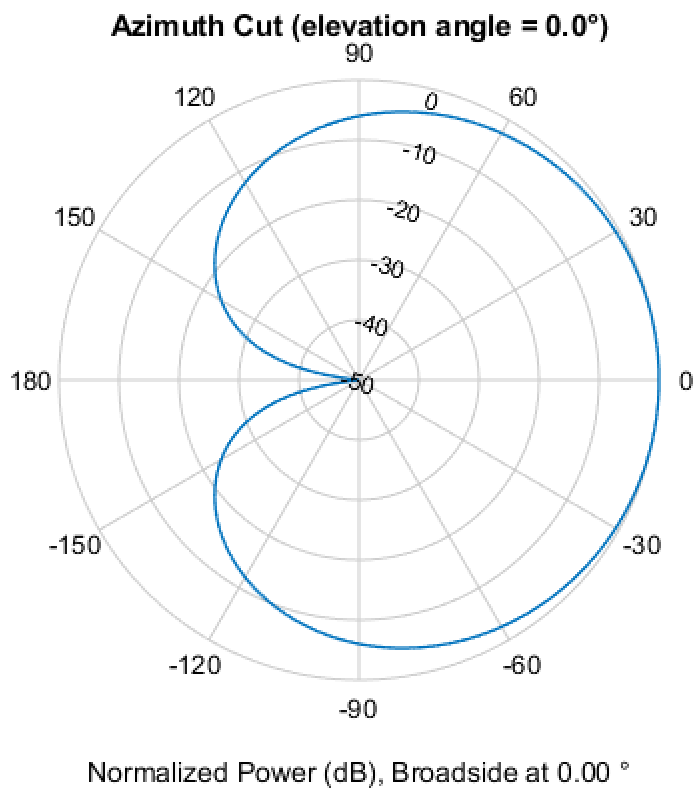
Obtain the microphone response in two directions.

```
resp = microphone([500 1500 2000],[0 0; 40 50]')
```

```
resp = 2×3

    1.0000    1.0000    1.0000
    0.7424    0.7939    0.7939
```

Display the microphone power pattern.

```
pattern(microphone,500,-180:180,0,'Type','powerdb')
```



Directivity of Custom Microphone Element

Compute the directivity of a custom microphone element. Create a custom cardioid microphone, and plot the microphone's response at 700 Hz for elevations between -90 and +90 degrees.

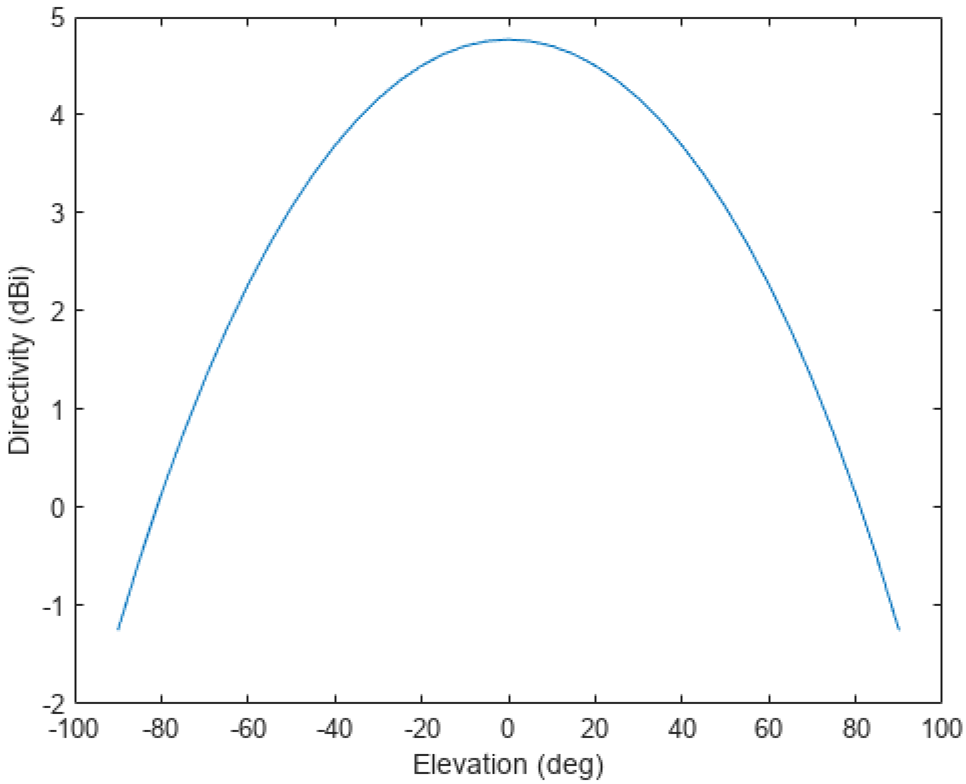
Define the pattern for the custom microphone element. The System object's `PolarPatternAngles` property has default value of `[-180:180]` degrees.

```
myAnt = phased.CustomMicrophoneElement;
myAnt.PolarPatternFrequencies = [500 1000];
myAnt.PolarPattern = mag2db([...
```

```
    0.5+0.5*cosd(myAnt.PolarPatternAngles);...
    0.6+0.4*cosd(myAnt.PolarPatternAngles)]);
```

Calculate the directivity as a function of elevation at zero degrees azimuth.

```
elev = [-90:5:90];
azm = zeros(size(elev));
ang = [azm;elev];
freq = 700;
d = directivity(myAnt,freq,ang);
plot(elev,d)
xlabel('Elevation (deg)')
ylabel('Directivity (dBi)')
```



The directivity is maximum at 0° elevation.

Custom Microphone Does Not Support Polarization

Show that the `phased.CustomMicrophoneElement` microphone element does not support polarization.

```
microphone = phased.CustomMicrophoneElement;
isPolarizationCapable(microphone)
```

```
ans = logical
   0
```

The returned value `0` shows that the custom microphone element does not support polarization.

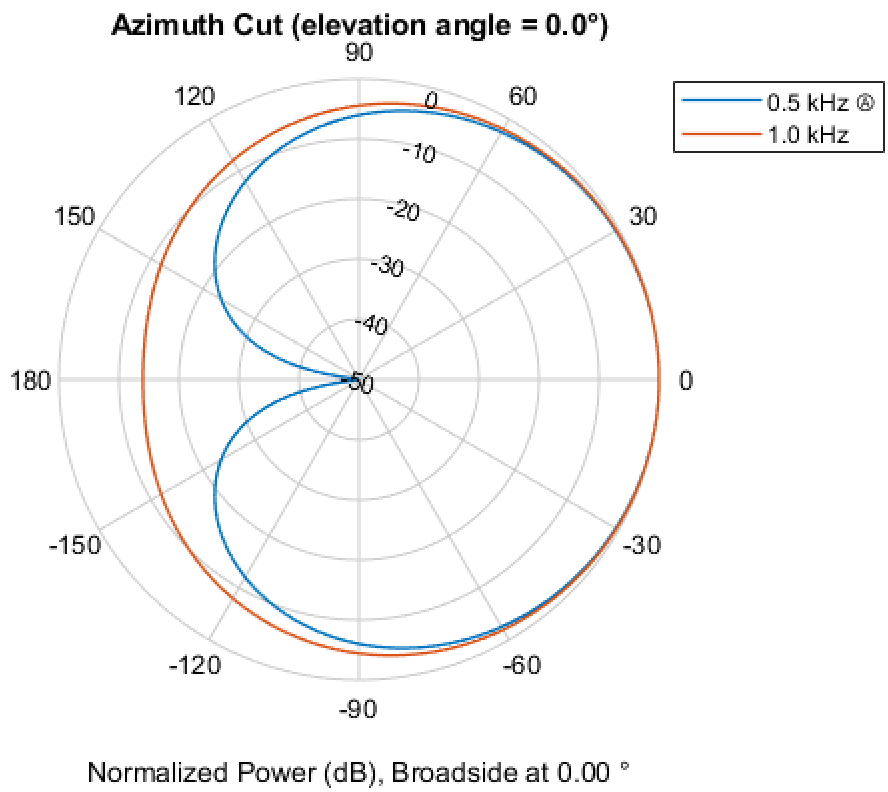
Azimuth Power Pattern and Directivity of Cardioid Microphone

Design a cardioid microphone to operate in the frequency range between 500 and 1000 Hz.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

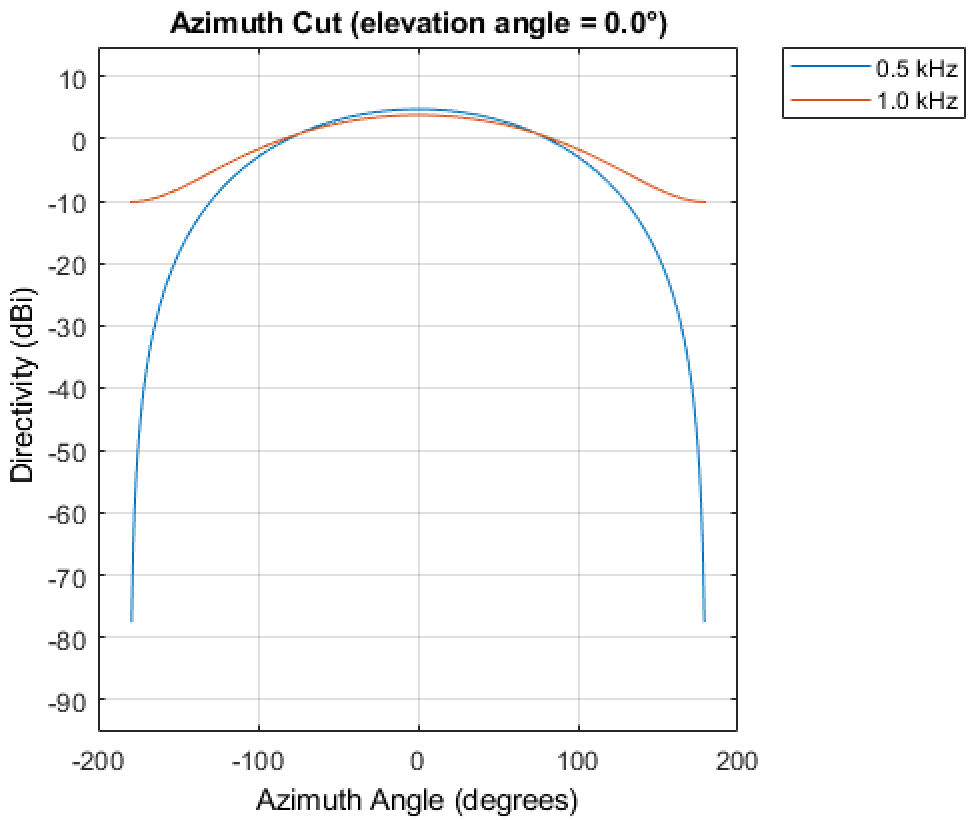
Display a polar plot of an azimuth cut of the response at 500 Hz and 1000 Hz.

```
fc = 500;
pattern(sCustMike,[fc 2*fc],[-180:180],0,...
    'CoordinateSystem','polar',...
    'Type','powerdb');
```



Plot the directivity as a line plot for the same two frequencies.

```
pattern(sCustMike,[fc 2*fc],[-180:180],0,...
    'CoordinateSystem','rectangular',...
    'Type','directivity');
```



Power Pattern of Cardioid Microphone in U/V Space

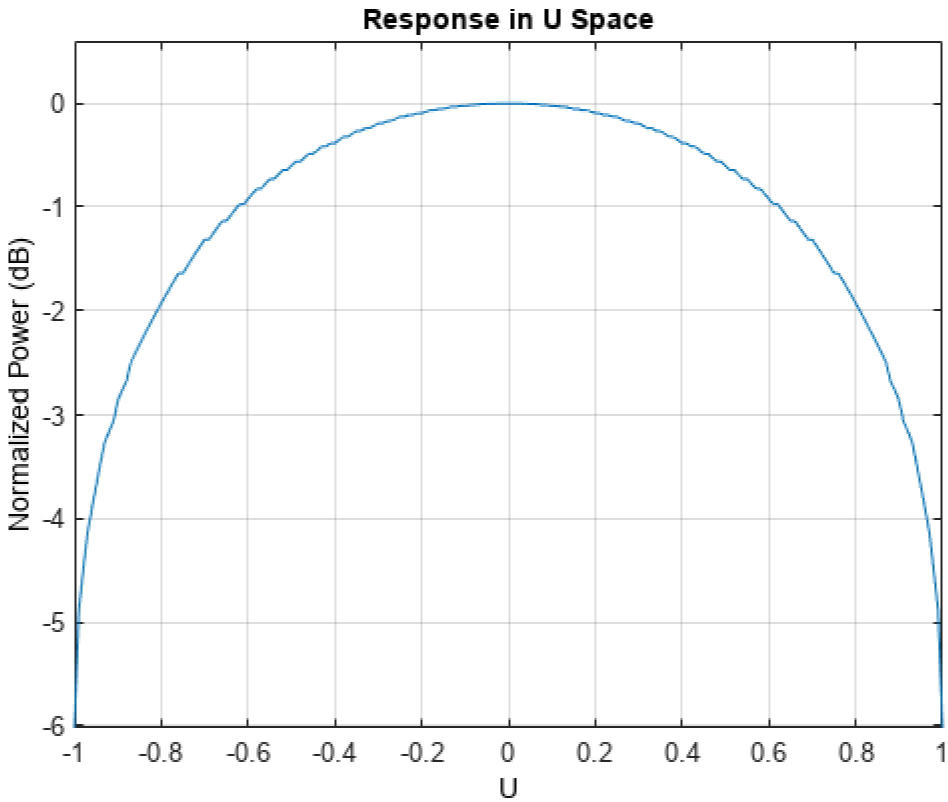
Plot a *u*-cut of the power pattern of a custom cardioid microphone designed to operate in the frequency range 500-1000 Hz.

Create a cardioid microphone.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

Plot the power pattern.

```
fc = 500;
pattern(sCustMike,fc,[-1:.01:1],0,...
    'CoordinateSystem','uv',...
    'Type','powerdb');
```



3-D Pattern of Cardioid Microphone over Restricted Range of Angles

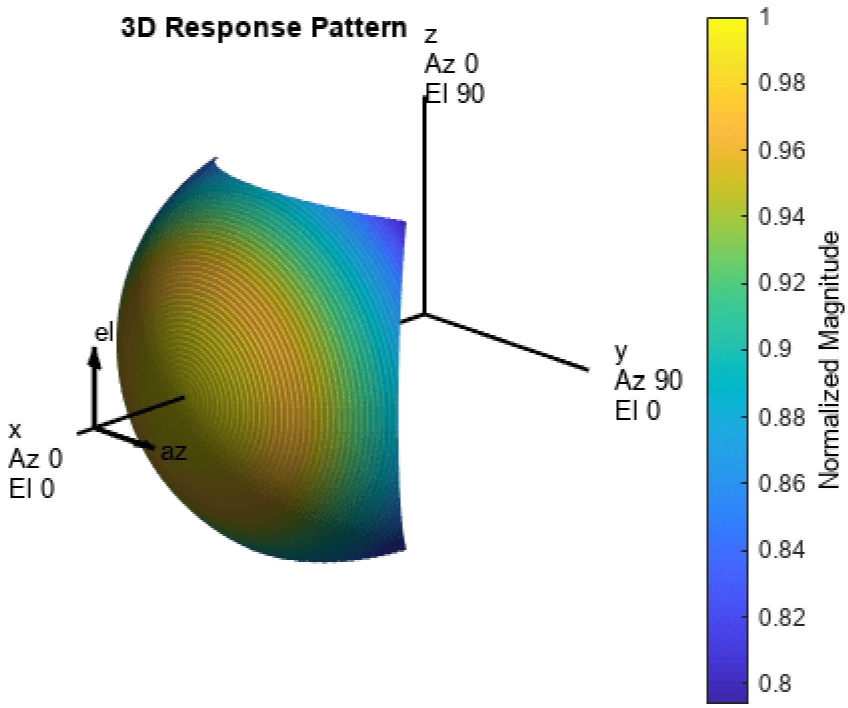
Plot the 3-D magnitude pattern of a custom cardioid microphone with both the azimuth and elevation angles restricted to the range -40 to 40 degrees in 0.1 degree increments.

Create a custom microphone element with a cardioid pattern.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

Plot the 3-D magnitude pattern.

```
fc = 500;
pattern(sCustMike,fc,[-40:0.1:40],[-40:0.1:40],...
    'CoordinateSystem','polar',...
    'Type','efield');
```



Azimuth Pattern of Cardioid Microphone over Reduced Angular Range

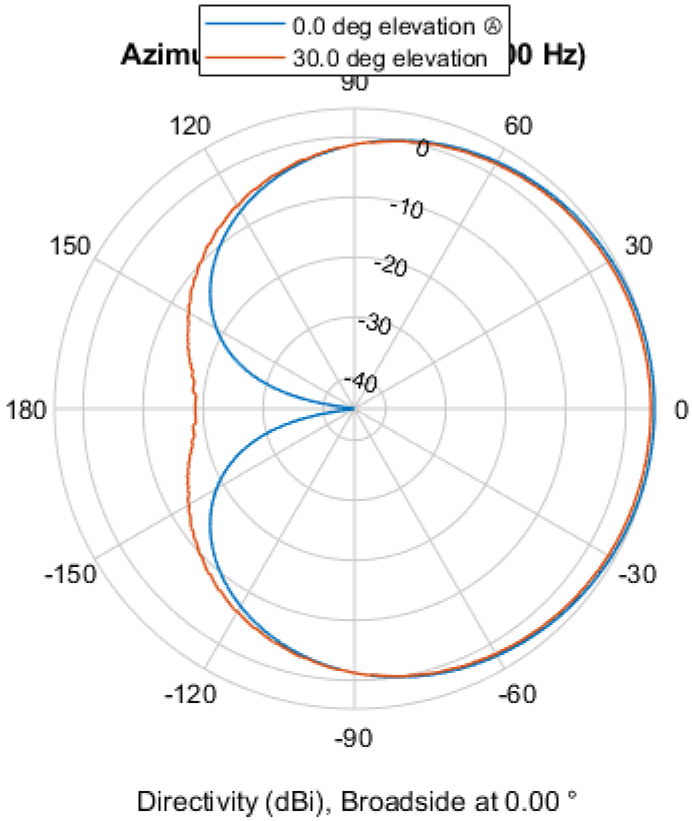
Plot the azimuth directivity pattern of a custom cardioid microphone at both 0 and 30 degrees elevation.

Create a custom microphone element with a cardioid pattern.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

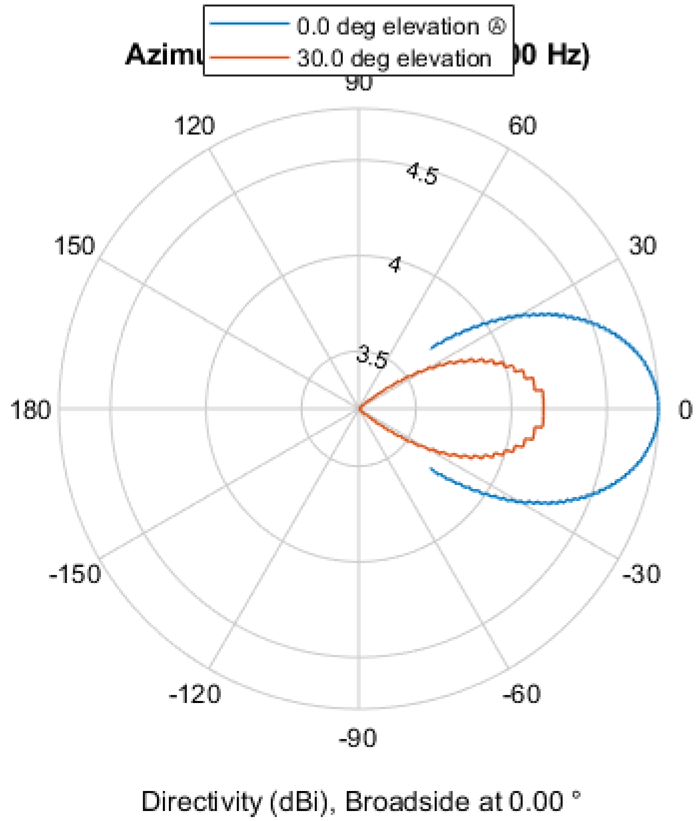
Plot the directivity at 500 Hz.

```
fc = 500;
patternAzimuth(sCustMike,fc,[0 30])
```



Plot the directivity for a reduced range of azimuth angles using the `Azimuth` parameter. Notice the change in scale.

```
fc = 500;
patternAzimuth(sCustMike,fc,[0 30],...
    'Azimuth',[-40:.1:40])
```



Elevation Pattern of Cardioid Microphone over Reduced Angular Range

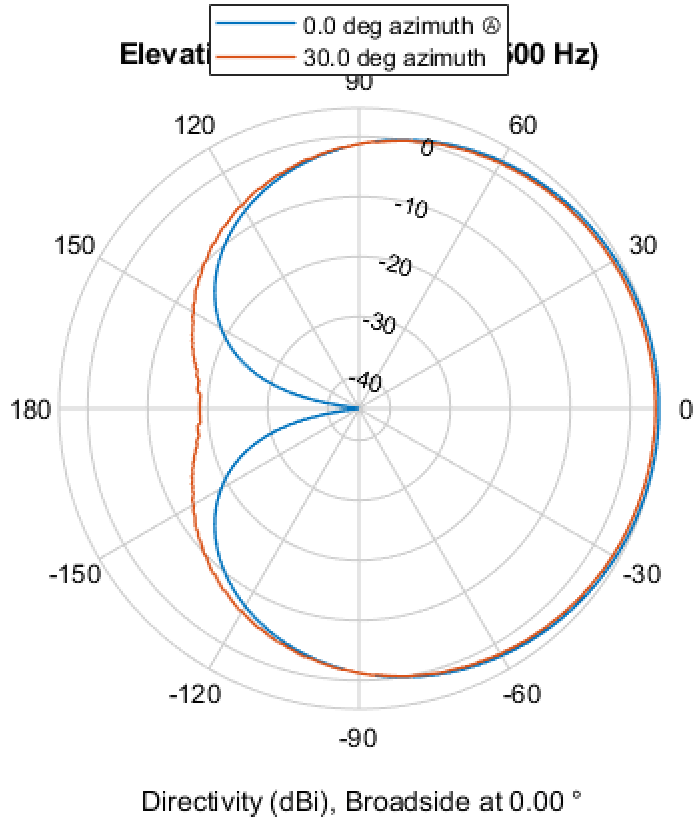
Plot the elevation directivity pattern of a custom cardioid microphone at both 0 and 45 degrees azimuth.

Create a custom microphone element with a cardioid pattern.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

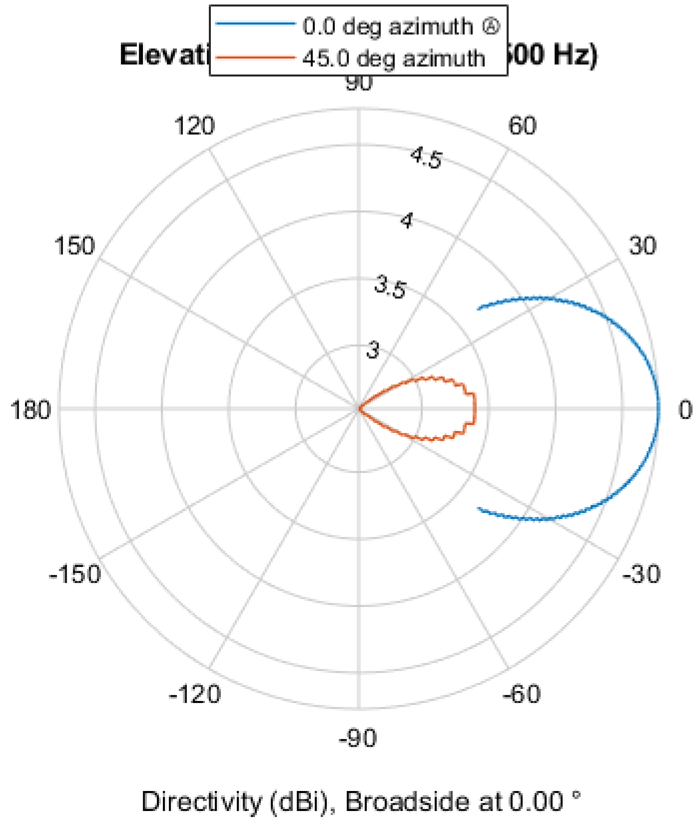
Plot the directivity at 500 Hz.

```
fc = 500;
patternElevation(sCustMike,fc,[0 30])
```



Plot the directivity for a reduced range of azimuth angles using the `Azimuth` parameter. Notice the change in scale.

```
fc = 500;
patternElevation(sCustMike,fc,[0 45],...
    'Elevation',[-40:.1:40])
```



Algorithms

The total response of a custom microphone element is a combination of its frequency response and spatial response. `phased.CustomMicrophoneElement` calculates both responses using nearest neighbor interpolation and then multiplies them to form the total response. When the `PolarPatternFrequencies` property value is nonscalar, the object specifies multiple polar patterns. In this case, the interpolation uses the polar pattern that is measured closest to the specified frequency.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, and `plotResponse` methods are not supported.

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.IsotropicProjector` | `phased.IsotropicHydrophone` | `phased.OmnidirectionalMicrophoneElement` | `phased.ULA` | `phased.URA` | `phased.ConformalArray` | `uv2azel` | `phitheta2azel`

directivity

System object: phased.CustomMicrophoneElement
Package: phased

Directivity of custom microphone element

Syntax

```
D = directivity(H,FREQ,ANGLE)
```

Description

D = directivity(H,FREQ,ANGLE) returns the "Directivity (dBi)" on page 1-347 of a custom microphone element, H, at frequencies specified by FREQ and in direction angles specified by ANGLE.

Input Arguments

H — Custom microphone element

System object

Custom microphone element specified as a phased.CustomMicrophoneElement System object.

Example: H = phased.CustomMicrophoneElement;

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.
- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: [1e8 2e6]

Data Types: double

ANGLE — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If ANGLE is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, [az;el]. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: `[45 60; 0 10]`

Data Types: `double`

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Custom Microphone Element

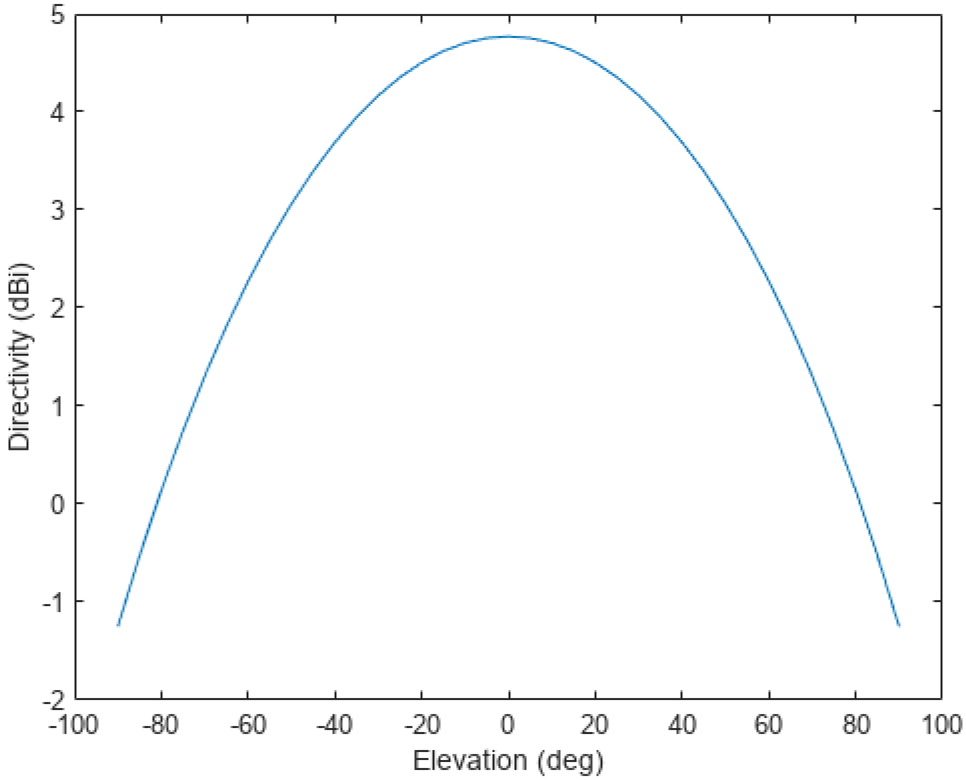
Compute the directivity of a custom microphone element. Create a custom cardioid microphone, and plot the microphone's response at 700 Hz for elevations between -90 and +90 degrees.

Define the pattern for the custom microphone element. The System object's `PolarPatternAngles` property has default value of `[-180:180]` degrees.

```
myAnt = phased.CustomMicrophoneElement;
myAnt.PolarPatternFrequencies = [500 1000];
myAnt.PolarPattern = mag2db([...
    0.5+0.5*cosd(myAnt.PolarPatternAngles);...
    0.6+0.4*cosd(myAnt.PolarPatternAngles)]);
```

Calculate the directivity as a function of elevation at zero degrees azimuth.

```
elev = [-90:5:90];
azm = zeros(size(elev));
ang = [azm;elev];
freq = 700;
d = directivity(myAnt,freq,ang);
plot(elev,d)
xlabel('Elevation (deg)')
ylabel('Directivity (dBi)')
```



The directivity is maximum at 0˚ elevation.

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also
pattern | patternElevation | patternAzimuth

isPolarizationCapable

System object: `phased.CustomMicrophoneElement`
Package: `phased`

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.CustomMicrophoneElement` supports polarization. An element supports polarization if it can create or respond to polarized fields. This microphone element, as with all microphone elements, does not support polarization.

Input Arguments

h — Custom microphone element

Custom microphone element specified as a `phased.CustomMicrophoneElement` System object.

Output Arguments

`flag` — Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the microphone element supports polarization or `false` if it does not. Because the `phased.CustomMicrophoneElement` object does not support polarization, `flag` is always returned as `false`.

Examples

Custom Microphone Does Not Support Polarization

Show that the `phased.CustomMicrophoneElement` microphone element does not support polarization.

```
microphone = phased.CustomMicrophoneElement;
isPolarizationCapable(microphone)
```

```
ans = logical
   0
```

The returned value `0` shows that the custom microphone element does not support polarization.

pattern

System object: phased.CustomMicrophoneElement
Package: phased

Plot custom microphone element directivity and patterns

Syntax

```
pattern(sElem,FREQ)
pattern(sElem,FREQ,AZ)
pattern(sElem,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sElem,FREQ)` plots the 3-D array directivity pattern (in dBi) for the element specified in `sElem`. The operating frequency is specified in `FREQ`.

`pattern(sElem,FREQ,AZ)` plots the element directivity pattern at the specified azimuth angle.

`pattern(sElem,FREQ,AZ,EL)` plots the element directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the element pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See “Convert plotResponse to pattern” on page 1-357 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sElem` — Custom microphone element

System object

Custom microphone element, specified as a `phased.CustomMicrophoneElement` System object.

Example: `sElem = phased.CustomMicrophoneElement;`

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the

pattern azimuth and elevation, respectively. AZ values must lie between –180° and 180°. EL values must lie between –90° and 90°. If 'CoordinateSystem' is set to 'uv', AZ and EL then specify *U* and *V* coordinates, respectively. AZ and EL must lie between -1 and 1.

Example: 'uv'

Data Types: char

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

PlotStyle — Plotting style
'overlay' (default) | 'waterfall'

Plotting style, specified as the comma-separated pair consisting of 'Plotstyle' and either 'overlay' or 'waterfall'. This parameter applies when you specify multiple frequencies in FREQ in 2-D plots. You can draw 2-D plots by setting one of the arguments AZ or EL to a scalar.

Data Types: char

Output Arguments

PAT — Element pattern
N-by-*M* real-valued matrix

Element pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of PAT correspond to the azimuth angles in the vector specified by EL_ANG. The columns correspond to the elevation angles in the vector specified by AZ_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

Examples

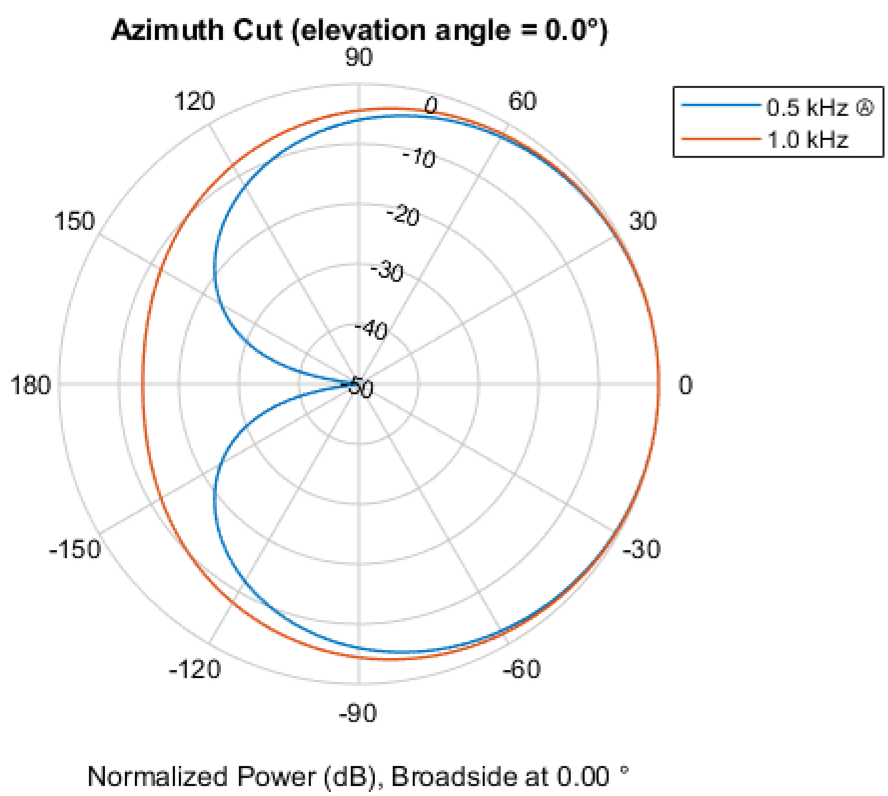
Azimuth Power Pattern and Directivity of Cardioid Microphone

Design a cardioid microphone to operate in the frequency range between 500 and 1000 Hz.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

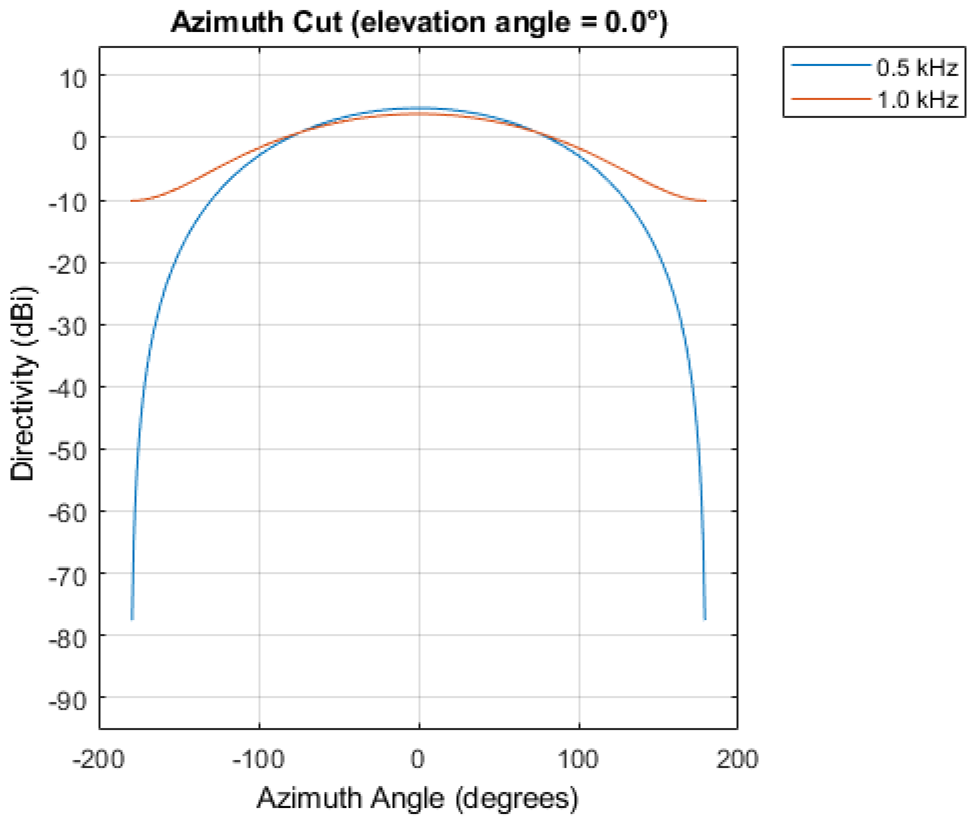
Display a polar plot of an azimuth cut of the response at 500 Hz and 1000 Hz.

```
fc = 500;
pattern(sCustMike,[fc 2*fc],[-180:180],0,...
    'CoordinateSystem','polar',...
    'Type','powerdb');
```



Plot the directivity as a line plot for the same two frequencies.

```
pattern(sCustMike,[fc 2*fc],[-180:180],0,...
    'CoordinateSystem','rectangular',...
    'Type','directivity');
```



Power Pattern of Cardioid Microphone in U/V Space

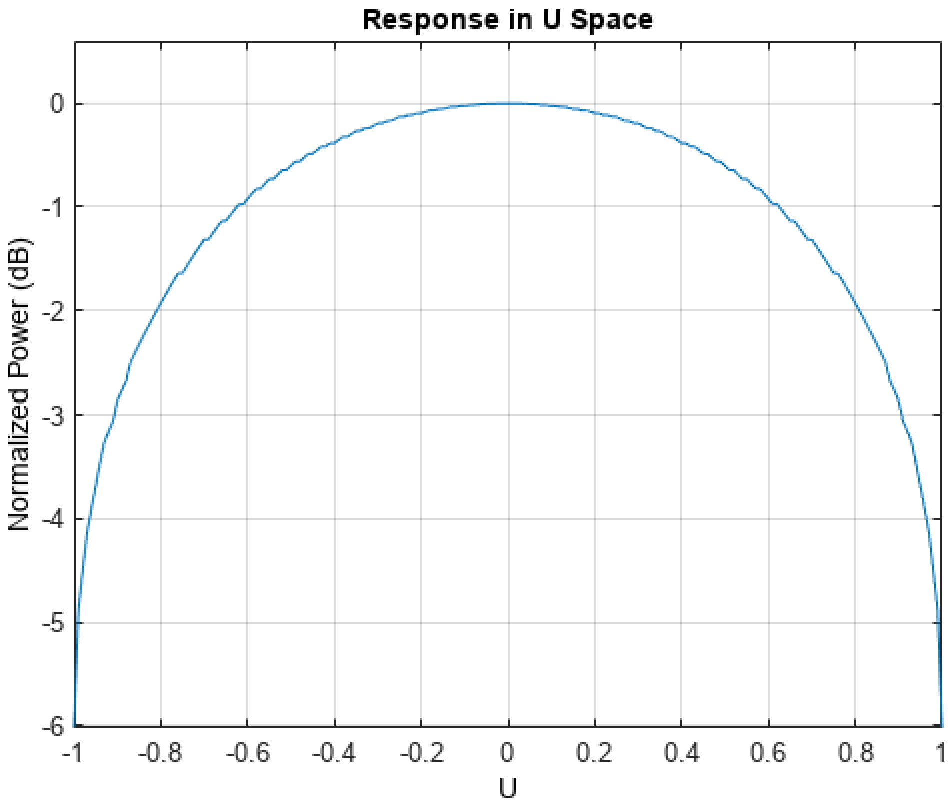
Plot a *u*-cut of the power pattern of a custom cardioid microphone designed to operate in the frequency range 500-1000 Hz.

Create a cardioid microphone.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

Plot the power pattern.

```
fc = 500;
pattern(sCustMike,fc,[-1:.01:1],0,...
    'CoordinateSystem','uv',...
    'Type','powerdb');
```



3-D Pattern of Cardioid Microphone over Restricted Range of Angles

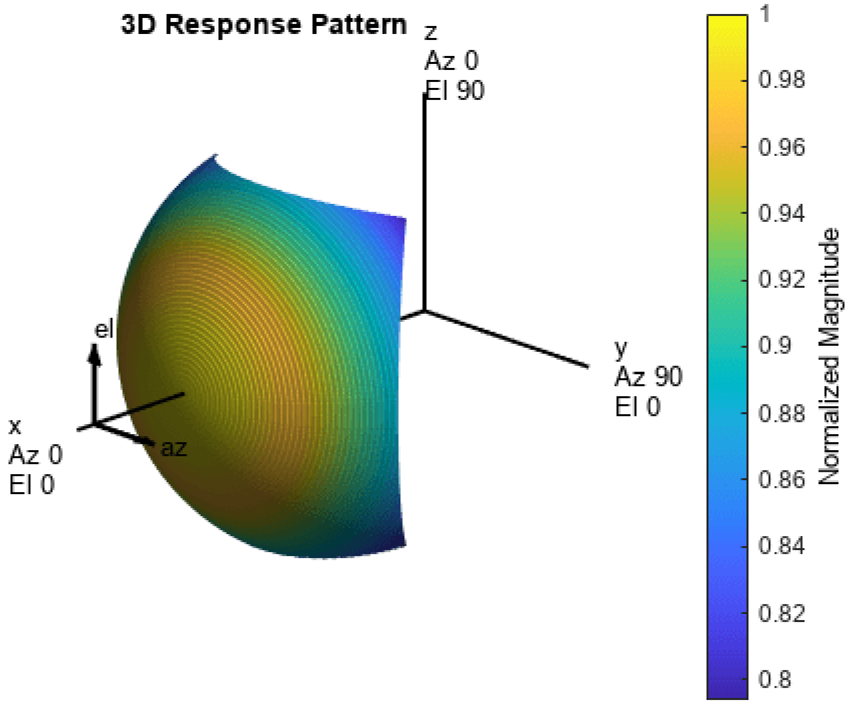
Plot the 3-D magnitude pattern of a custom cardioid microphone with both the azimuth and elevation angles restricted to the range -40 to 40 degrees in 0.1 degree increments.

Create a custom microphone element with a cardioid pattern.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

Plot the 3-D magnitude pattern.

```
fc = 500;
pattern(sCustMike,fc,[-40:0.1:40],[-40:0.1:40],...
    'CoordinateSystem','polar',...
    'Type','efield');
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.

Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.

Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.

'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the UV grid values using AZ and EL.
</td>
</tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
	Display space | | 'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.	
'CutAngle' name-value pair	Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.	No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.
'NormalizeResponse' name-value pair	Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.	Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.
'OverlayFreq' name-value pair	Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.	'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot. The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.
'Polarization' name-value pair	Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.	'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.
'Unit' name-value pair	Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.	'Type' name-value pair, uses equivalent options with different names **plotResponse** — **pattern** 'db' — 'powerdb' 'mag' — 'efield' 'pow' — 'power' 'dbi' — 'directivity'
'Weights' name-value pair	Array element tapers (or weights).	'Weights' name-value pair (no change).
'AzimuthAngles' name-value pair	Azimuth angles used to display the antenna or array response.	AZ argument

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `patternElevation`

patternAzimuth

System object: `phased.CustomMicrophoneElement`
Package: `phased`

Plot custom microphone element directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sElem,FREQ)
patternAzimuth(sElem,FREQ,EL)
patternAzimuth(sElem,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sElem,FREQ)` plots the 2-D element directivity pattern versus azimuth (in dBi) for the element `sElem` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

`patternAzimuth(sElem,FREQ,EL)`, in addition, plots the 2-D element directivity pattern versus azimuth (in dBi) at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sElem,FREQ,EL,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sElem` — Custom microphone element
System object

Custom microphone element, specified as a `phased.CustomMicrophoneElement` System object.

Example: `sElem = phased.CustomMicrophoneElement;`

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

Examples

Azimuth Pattern of Cardioid Microphone over Reduced Angular Range

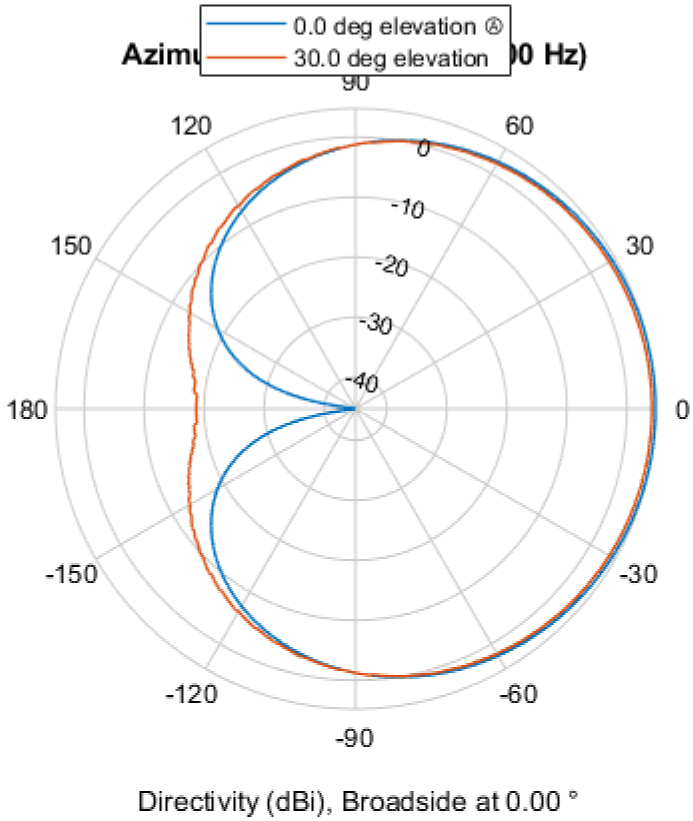
Plot the azimuth directivity pattern of a custom cardioid microphone at both 0 and 30 degrees elevation.

Create a custom microphone element with a cardioid pattern.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

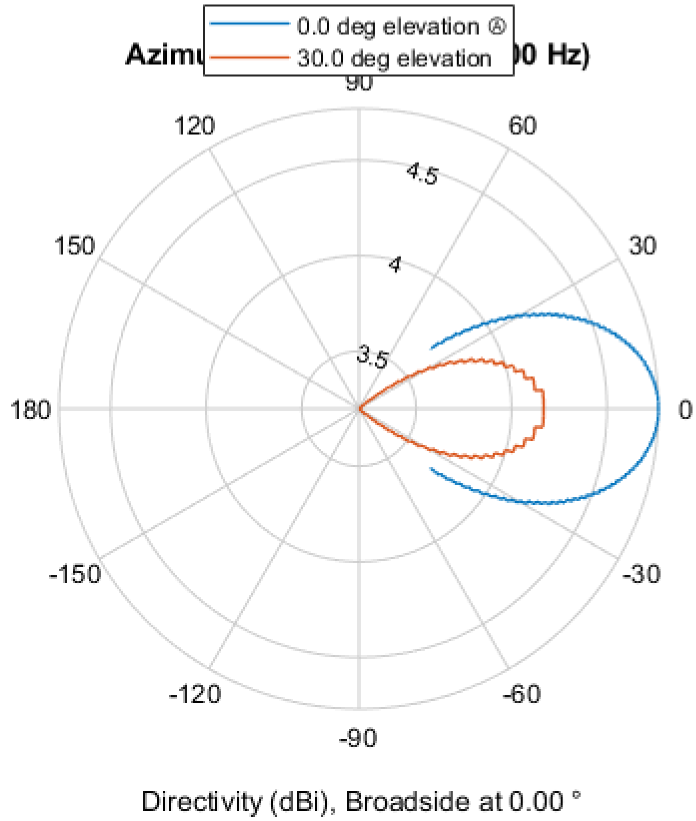
Plot the directivity at 500 Hz.

```
fc = 500;
patternAzimuth(sCustMike,fc,[0 30])
```



Plot the directivity for a reduced range of azimuth angles using the `Azimuth` parameter. Notice the change in scale.

```
fc = 500;
patternAzimuth(sCustMike,fc,[0 30],...
    'Azimuth',[-40:.1:40])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: `phased.CustomMicrophoneElement`
Package: `phased`

Plot custom microphone element directivity or pattern versus elevation

Syntax

```
patternElevation(sElem,FREQ)
patternElevation(sElem,FREQ,AZ)
patternElevation(sElem,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sElem,FREQ)` plots the 2-D element directivity pattern versus elevation (in dBi) for the element `sElem` at zero degrees azimuth angle. The argument `FREQ` specifies the operating frequency.

`patternElevation(sElem,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sElem,FREQ,AZ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sElem` — Custom microphone element
System object

Custom microphone element, specified as a `phased.CustomMicrophoneElement` System object.

Example: `sElem = phased.CustomMicrophoneElement;`

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

Elevation — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Elevation Pattern of Cardioid Microphone over Reduced Angular Range

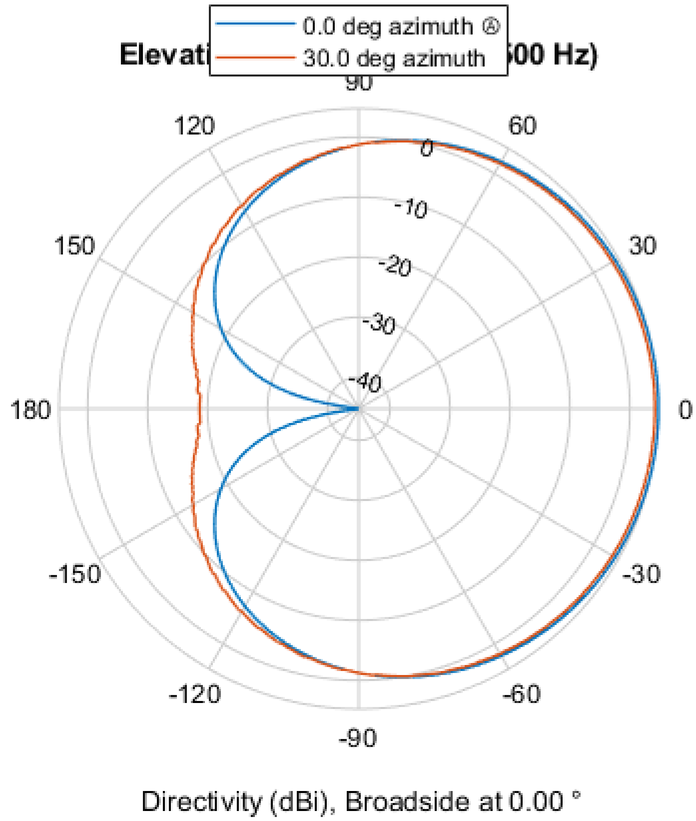
Plot the elevation directivity pattern of a custom cardioid microphone at both 0 and 45 degrees azimuth.

Create a custom microphone element with a cardioid pattern.

```
sCustMike = phased.CustomMicrophoneElement;
sCustMike.PolarPatternFrequencies = [500 1000];
sCustMike.PolarPattern = mag2db([...
    0.5+0.5*cosd(sCustMike.PolarPatternAngles);...
    0.6+0.4*cosd(sCustMike.PolarPatternAngles)]);
```

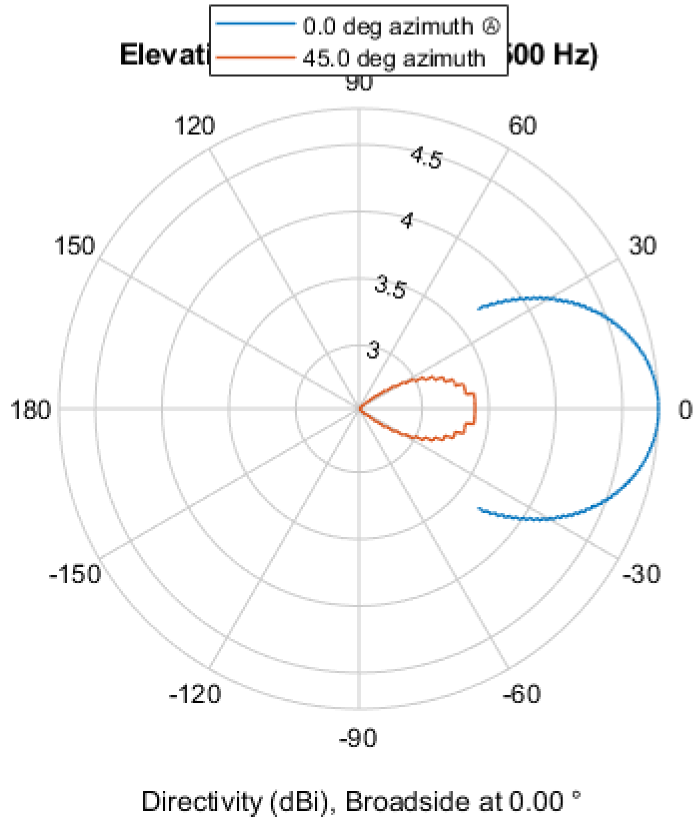
Plot the directivity at 500 Hz.

```
fc = 500;
patternElevation(sCustMike,fc,[0 30])
```



Plot the directivity for a reduced range of azimuth angles using the `Azimuth` parameter. Notice the change in scale.

```
fc = 500;
patternElevation(sCustMike,fc,[0 45],...
    'Elevation',[-40:.1:40])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth`

plotResponse

System object: `phased.CustomMicrophoneElement`
Package: `phased`

Plot response pattern of microphone

Syntax

```
plotResponse(H,FREQ)
plotResponse(H,FREQ,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ)` plots the element response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`.

`plotResponse(H,FREQ,Name,Value)` plots the element response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

`H`

Element System object

`FREQ`

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CutAngle`

Cut angle specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

Format

Format of the plot, using one of 'Line', 'Polar', or 'UV'. If you set Format to 'UV', FREQ must be a scalar.

Default: 'Line'

NormalizeResponse

Set this value to true to normalize the response pattern. Set this value to false to plot the response pattern without normalizing it. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: true

OverlayFreq

Set this value to true to overlay pattern cuts in a 2-D line plot. Set this value to false to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is false, FREQ must be a vector with at least two entries.

This parameter applies only when Format is not 'Polar' and RespCut is not '3D'.

Default: true

Polarization

Specify the polarization options for plotting the antenna response pattern. The allowable values are |'None' | 'Combined' | 'H' | 'V' | where

- 'None' specifies plotting a nonpolarized response pattern
- 'Combined' specifies plotting a combined polarization response pattern
- 'H' specifies plotting the horizontal polarization response pattern
- 'V' specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is 'None'. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: 'None'

RespCut

Cut of the response. Valid values depend on Format, as follows:

- If Format is 'Line' or 'Polar', the valid values of RespCut are 'Az', 'El', and '3D'. The default is 'Az'.
- If Format is 'UV', the valid values of RespCut are 'U' and '3D'. The default is 'U'.

If you set RespCut to '3D', FREQ must be a scalar.

Unit

The unit of the plot. Valid values are 'db', 'mag', 'pow', or 'dbi'. This parameter determines the type of plot that is produced.

Unit value	Plot type
db	power pattern in dB scale
mag	field pattern
pow	power pattern
dbi	directivity

Default: 'db'

AzimuthAngles

Azimuth angles for plotting element response, specified as a row vector. The AzimuthAngles parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'Az' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the AzimuthAngles and ElevationAngles parameters simultaneously.

Default: [-180:180]

ElevationAngles

Elevation angles for plotting element response, specified as a row vector. The ElevationAngles parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'El' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the ElevationAngles and AzimuthAngles parameters simultaneously.

Default: [-90:90]

UGrid

U coordinate values for plotting element response, specified as a row vector. The UGrid parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to 'U' or '3D'. The values of UGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the UGrid and VGrid parameters simultaneously.

Default: [-1:0.01:1]

VGrid

V coordinate values for plotting element response, specified as a row vector. The VGrid parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to '3D'. The values of VGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the VGrid and UGrid parameters simultaneously.

Default: [-1:0.01:1]

Examples

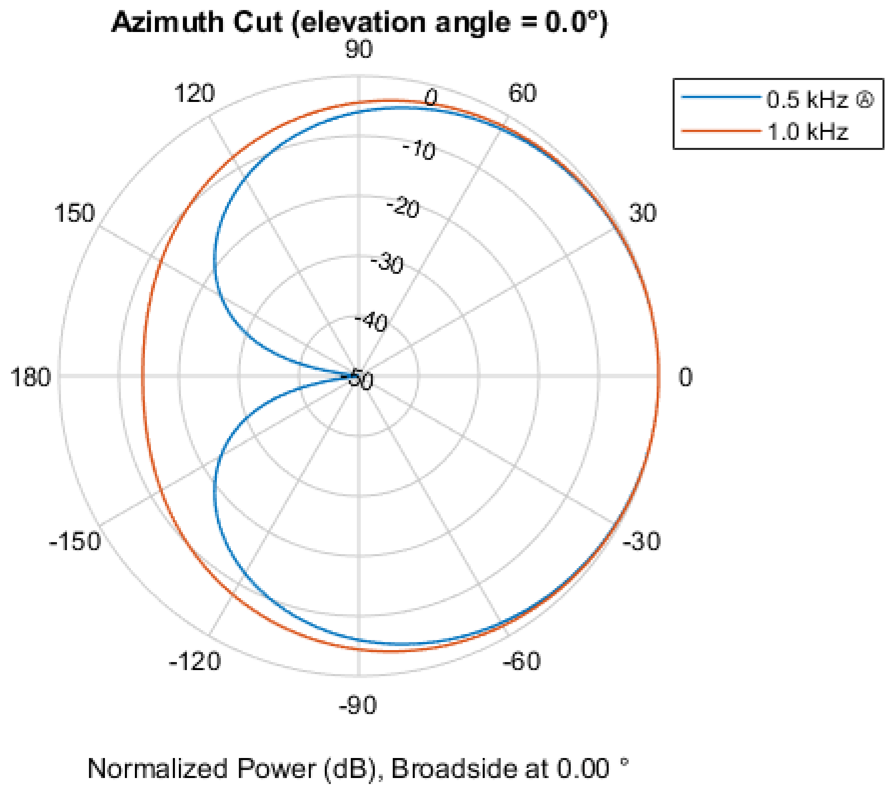
Azimuth Response and Directivity of Cardioid Microphone

Design a cardioid microphone to operate in the frequency range between 500 and 1000 Hz.

```
h = phased.CustomMicrophoneElement;
h.PolarPatternFrequencies = [500 1000];
h.PolarPattern = mag2db([...
    0.5+0.5*cosd(h.PolarPatternAngles);...
    0.6+0.4*cosd(h.PolarPatternAngles)]);
```

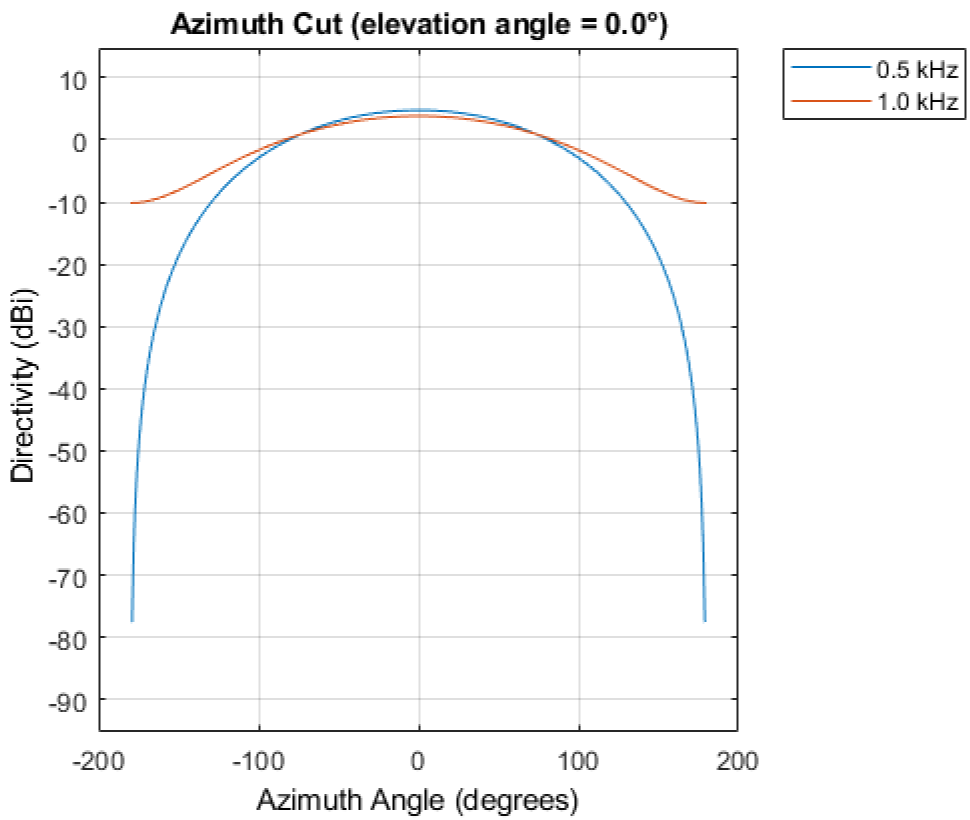
Display a polar plot of an azimuth cut of the response at 500 Hz and 1000 Hz.

```
fc = 500;
plotResponse(h,[fc 2*fc],'RespCut','Az','Format','Polar');
```



Plot the directivity as a line plot for the same two frequencies.

```
plotResponse(h,[fc 2*fc],'RespCut','Az','Format','Line','Unit','dbi');
```



Response of Cardioid Microphone in U/V Space

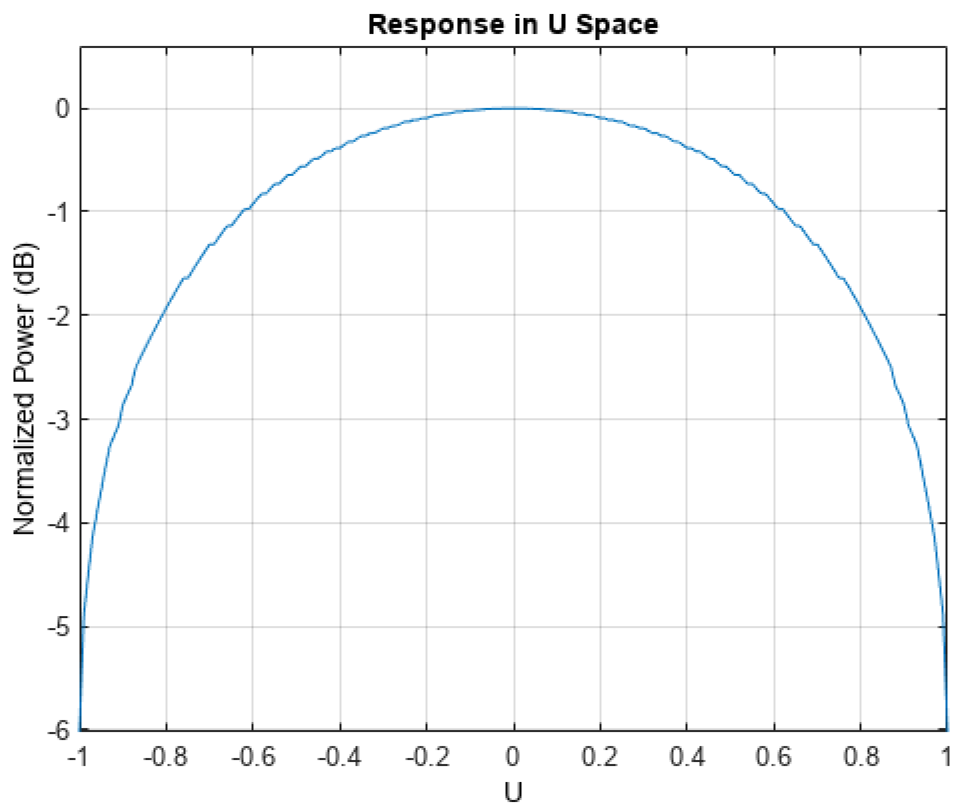
Plot a *u*-cut of the response of a custom cardioid microphone that is designed to operate in the frequency range 500-1000 Hz.

Create a cardioid microphone.

```
h = phased.CustomMicrophoneElement;
h.PolarPatternFrequencies = [500 1000];
h.PolarPattern = mag2db([...
    0.5+0.5*cosd(h.PolarPatternAngles);...
    0.6+0.4*cosd(h.PolarPatternAngles)]);
```

Plot the response.

```
fc = 500;
plotResponse(h,fc,'Format','UV');
```



3-D Response of Cardioid Microphone over Restricted Range of Angles

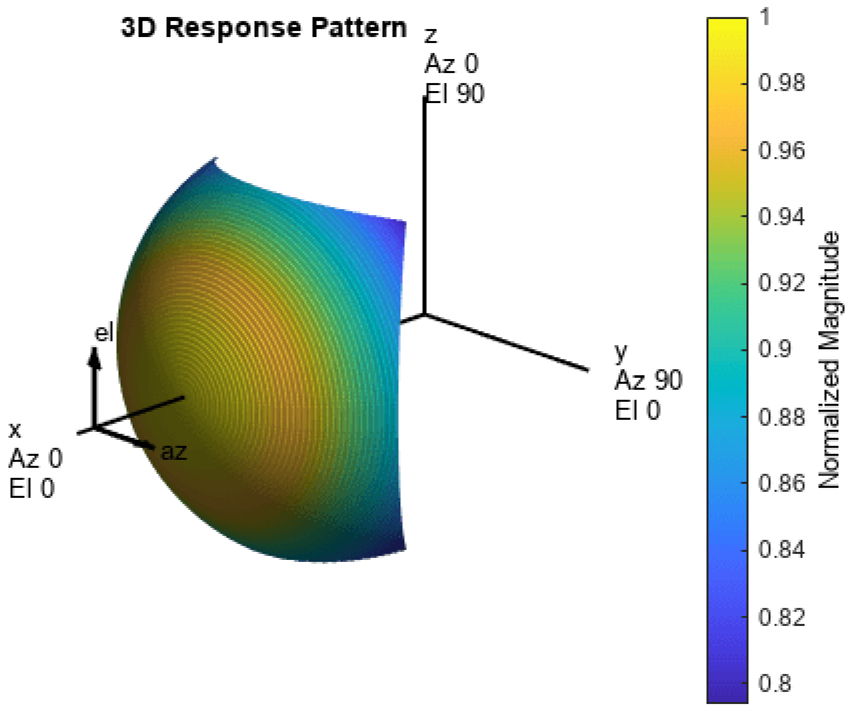
Plot the 3-D response of a custom cardioid microphone in space but with both the azimuth and elevation angles restricted to the range -40 to 40 degrees in 0.1 degree increments.

Create a custom microphone element with a cardioid pattern.

```
h = phased.CustomMicrophoneElement;
h.PolarPatternFrequencies = [500 1000];
h.PolarPattern = mag2db([...
    0.5+0.5*cosd(h.PolarPatternAngles);...
    0.6+0.4*cosd(h.PolarPatternAngles)]);
```

Plot the 3-D response.

```
fc = 500;
plotResponse(h,fc,'Format','polar','RespCut','3D',...
    'Unit','mag','AzimuthAngles',[-40:0.1:40],...
    'ElevationAngles',[-40:0.1:40]);
```



See Also

`uv2azel` | `azel2uv`

step

System object: `phased.CustomMicrophoneElement`
Package: `phased`

Output response of microphone

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the microphone's magnitude response, `RESP`, at frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`H`

Microphone object.

`FREQ`

Frequencies in hertz. `FREQ` is a row vector of length L.

`ANG`

Directions in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If `ANG` is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

Output Arguments

RESP

Response of microphone. RESP is an M-by-L matrix that contains the responses of the microphone element at the M angles specified in ANG and the L frequencies specified in FREQ.

Examples

Custom Microphone Response

Construct a custom cardioid microphone with an operating frequency of 700 Hz. Find the microphone response in the directions: *(0,0)* degrees azimuth and elevation and *(40,50)* degrees azimuth and elevation.

```
microphone = phased.CustomMicrophoneElement;
microphone.PolarPatternFrequencies = [500 1000];
microphone.PolarPattern = mag2db([ ...
    0.5+0.5*cosd(microphone.PolarPatternAngles); ...
    0.6+0.4*cosd(microphone.PolarPatternAngles)]);
fc = 700;
ang = [0 0; 40 50]';
resp = microphone(fc,ang)
```

```
resp = 2×1

    1.0000
    0.7424
```

Algorithms

The total response of a custom microphone element is a combination of its frequency response and spatial response. `phased.CustomMicrophoneElement` calculates both responses using nearest neighbor interpolation and then multiplies them to form the total response. When the `PolarPatternFrequencies` property value is nonscalar, the object specifies multiple polar patterns. In this case, the interpolation uses the polar pattern that is measured closest to the specified frequency.

See Also

`uv2azel` | `phitheta2azel`

phased.DopplerEstimator

Package: `phased`

Doppler estimation

Description

The `phased.DopplerEstimator` System object estimates Doppler frequencies of targets. Input to the estimator consists of detection locations output from a detector, and a range-Doppler response data cube. When detections are clustered, the Doppler frequencies are computed using cluster information. Clustering associates multiple detections into one extended detection.

To compute Doppler values for detections:

1. Define and set up your Doppler estimator using the "Construction" on page 1-382 procedure that follows.
2. Call the `step` method to compute the Doppler of detections, using the properties you specify for the `phased.DopplerEstimator` System object.

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`estimator = phased.DopplerEstimator` creates a Doppler estimator System object, `estimator`.

`estimator = phased.DopplerEstimator(Name,Value)` creates a System object, `estimator`, with each specified property `Name` set to the specified `Value`. You can specify additional name and value pair arguments in any order as (`Name1,Value1`,...,`NameN,ValueN`).

Properties

`NumEstimatesSource` — Source of requested number of Doppler estimates
`'Auto'` (default) | `'Property'`

Source of the number of requested Doppler estimates, specified as `'Auto'` or `'Property'`.

If you set this property to `'Auto'`, the number of estimates equals the number of columns in the `detidx` input argument of the `step` method. If cluster IDs are provided, the number of estimates equals the number of unique cluster IDs.

If you set this property to `'Property'`, the number of reported estimates is obtained from the value of the `NumEstimates` property.

Data Types: `char`

`NumEstimates` — Maximum number of estimates
`1` (default) | positive integer

The maximum number of estimates to report, specified as a positive integer. When the number of requested estimates is greater than the number of columns in the `detidx` argument of the `step` method, the remainder is filled with `NaN`.

Dependencies

To enable this property, set the `NumEstimatesSource` property to `'Property'`.

Data Types: `c` | `double`

`ClusterInputPort` — Accept `clusterids` as input
`false` (default) | `true`

Option to accept `clusterids` as an input argument to the `step` method, specified as `false` or `true`. Setting this property to `true` enables the `clusterid` input argument of the `step` method.

Data Types: `logical`

`VarianceOutputPort` — Enable output of Doppler variance estimates
`false` (default) | `true`

Option to enable output of Doppler variance estimate, specified as `false` or `true`. Doppler variances estimates are returned in the `dopvar` output argument of the `step` method.

Data Types: `logical`

`NumPulses` — Number of pulses in Doppler-processed waveform
`2` (default) | positive integer

The number of pulses in the Doppler processed data cube, specified as a positive integer.

Dependencies

To enable this property, set the `VarianceOutputPort` property to `true`.

Data Types: `single` | `double`

`NoisePowerSource` — Source of noise power values
`'Property'` (default) | `'Input port'`

Source of noise power values, specified as `'Property'` or `'Input port'`. Noise power is used to compute Doppler estimation variance and SNR. If you set this property to `'Property'`, the value of the `NoisePower` property represents the noise power at the detection locations. If you set this property to `'Input port'`, you can specify noise power using the `noisepower` input argument of the `step` method.

Data Types: `char`

`NoisePower` — Noise power
`1.0` (default) | positive scalar

Constant noise power value over the range-Doppler data cube, specified as a positive scalar. Noise power units are linear. The same noise power value is applied to all detections.

Dependencies

To enable this property, set the `VarianceOutputPort` property to `true` and set `NoisePowerSource` to `'Property'`.

Data Types: `single` | `double`

Methods

step	Estimate target Doppler

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Estimate Range and Speed of Three Targets

To estimate the range and speed of three targets, create a range-Doppler map using the `phased.RangeDopplerResponse` System object™. Then use the `phased.RangeEstimator` and `phased.DopplerEstimator` System objects to estimate range and speed. The transmitter and receiver are collocated isotropic antenna elements forming a monostatic radar system.

The transmitted signal is a linear FM waveform with a pulse repetition interval (PRI) of 7.0 μs and a duty cycle of 2%. The operating frequency is 77 GHz and the sample rate is 150 MHz.

```
fs = 150e6;
c = physconst('LightSpeed');
fc = 77.0e9;
pri = 7e-6;
prf = 1/pri;
```

Set up the scenario parameters. The transmitter and receiver are stationary and located at the origin. The targets are 500, 530, and 750 meters from the radar along the *x*-axis. The targets move along the *x*-axis at speeds of –60, 20, and 40 m/s. All three targets have a nonfluctuating radar cross-section (RCS) of 10 dB. Create the target and radar platforms.

```
Numtgts = 3;
tgtpos = zeros(Numtgts);
tgtpos(1,:) = [500 530 750];
tgtvel = zeros(3,Numtgts);
tgtvel(1,:) = [-60 20 40];
tgtrcs = db2pow(10)*[1 1 1];
tgtmotion = phased.Platform(tgtpos,tgtvel);
target = phased.RadarTarget('PropagationSpeed',c,'OperatingFrequency',fc, ...
    'MeanRCS',tgtrcs);
radarpos = [0;0;0];
radarvel = [0;0;0];
radarmotion = phased.Platform(radarpos,radarvel);
```

Create the transmitter and receiver antennas.

```
txantenna = phased.IsotropicAntennaElement;
rxantenna = clone(txantenna);
```

Set up the transmitter-end signal processing. Create an upsweep linear FM signal with a bandwidth of one half the sample rate. Find the length of the PRI in samples and then estimate the rms bandwidth and range resolution.

```
bw = fs/2;
waveform = phased.LinearFMWaveform('SampleRate',fs, ...
    'PRF',prf,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',fs/2, ...
    'DurationSpecification','Duty cycle','DutyCycle',0.02);
sig = waveform();
Nr = length(sig);
bwrms = bandwidth(waveform)/sqrt(12);
rngrms = c/bwrms;
```

Set up the transmitter and radiator System object properties. The peak output power is 10 W and the transmitter gain is 36 dB.

```
peakpower = 10;
txgain = 36.0;
transmitter = phased.Transmitter( ...
    'PeakPower',peakpower, ...
    'Gain',txgain, ...
    'InUseOutputPort',true);
radiator = phased.Radiator( ...
    'Sensor',txantenna,...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc);
```

Set up the free-space channel in two-way propagation mode.

```
channel = phased.FreeSpace( ...
    'SampleRate',fs, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc, ...
    'TwoWayPropagation',true);
```

Set up the receiver-end processing. Set the receiver gain and noise figure.

```
collector = phased.Collector( ...
    'Sensor',rxantenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc);
rxgain = 42.0;
noisefig = 1;
receiver = phased.ReceiverPreamp( ...
    'SampleRate',fs, ...
    'Gain',rxgain, ...
    'NoiseFigure',noisefig);
```

Loop over the pulses to create a data cube of 128 pulses. For each step of the loop, move the target and propagate the signal. Then put the received signal into the data cube. The data cube contains the received signal per pulse. Ordinarily, a data cube has three dimensions where the last dimension corresponds to antennas or beams. Because only one sensor is used, the cube has only two dimensions.

The processing steps are:

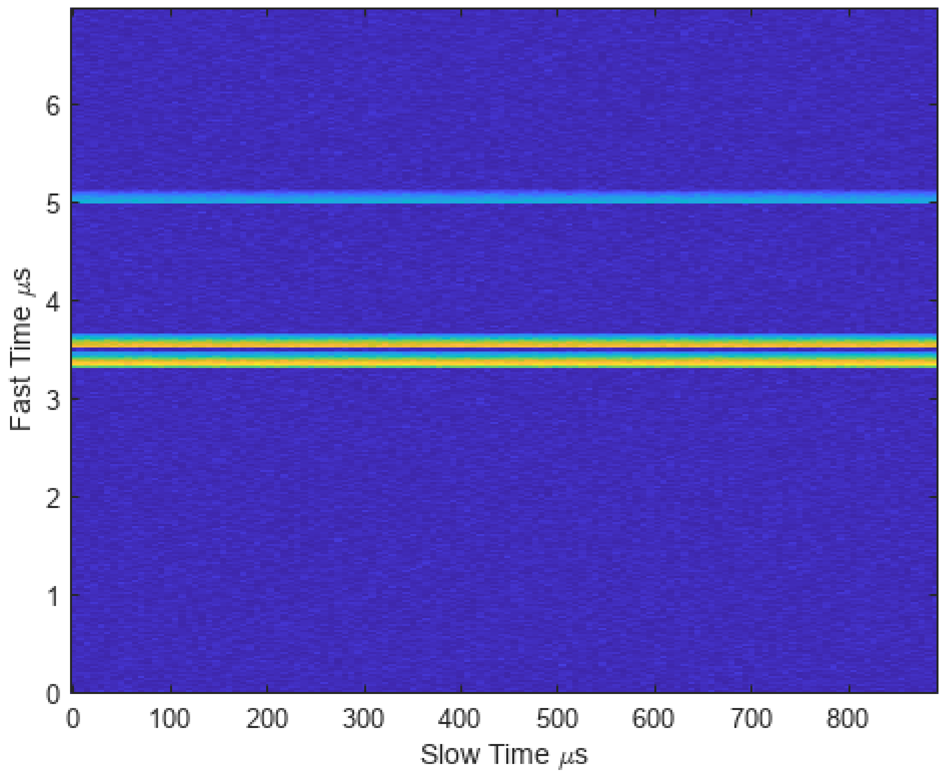
1. Move the radar and targets.

2. Transmit a waveform.
3. Propagate the waveform signal to the target.
4. Reflect the signal from the target.
5. Propagate the waveform back to the radar. Two-way propagation enables you to combine the return propagation with the outbound propagation.
6. Receive the signal at the radar.
7. Load the signal into the data cube.

```
Np = 128;
dt = pri;
cube = zeros(Nr,Np);
for n = 1:Np
    [sensorpos,sensorvel] = radarmotion(dt);
    [tgtpos,tgtvel] = tgtmotion(dt);
    [tgtrng,tgtang] = rangeangle(tgtpos,sensorpos);
    sig = waveform();
    [txsig,txstatus] = transmitter(sig);
    txsig = radiator(txsig,tgtang);
    txsig = channel(txsig,sensorpos,tgtpos,sensorvel,tgtvel);
    tgtsig = target(txsig);
    rxcol = collector(tgtsig,tgtang);
    rxsig = receiver(rxcol);
    cube(:,n) = rxsig;
end
```

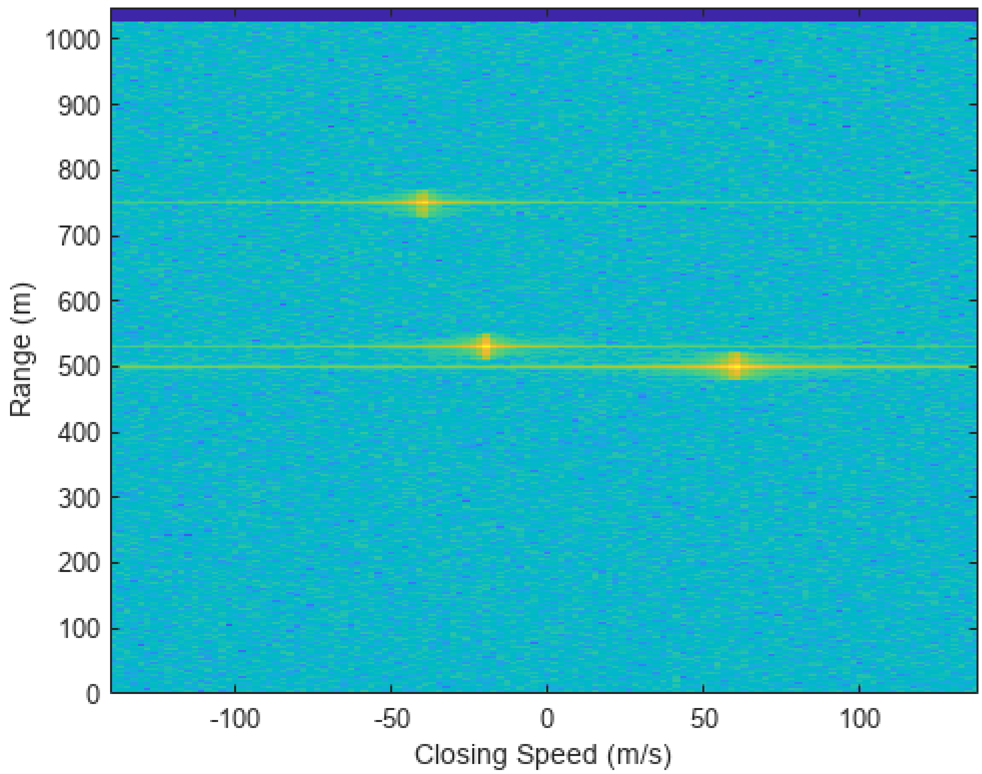
Display the data cube containing signals per pulse.

```
imagesc([0:(Np-1)]*pri*1e6,[0:(Nr-1)]/fs*1e6,abs(cube))
xlabel('Slow Time {\mu}s')
ylabel('Fast Time {\mu}s')
axis xy
```



Create and display the range-Doppler image for 128 Doppler bins. The image shows range vertically and speed horizontally. Use the linear FM waveform for match filtering. The image is here is the range-Doppler map.

```
ndop = 128;
rangedopresp = phased.RangeDopplerResponse('SampleRate',fs, ...
    'PropagationSpeed',c,'DopplerFFTLengthSource','Property', ...
    'DopplerFFTLength',ndop,'DopplerOutput','Speed', ...
    'OperatingFrequency',fc);
matchingcoeff = getMatchedFilter(waveform);
[rngdopresp,rnggrid,dopgrid] = rangedopresp(cube,matchingcoeff);
imagesc(dopgrid,rnggrid,10*log10(abs(rngdopresp)))
xlabel('Closing Speed (m/s)')
ylabel('Range (m)')
axis xy
```



Because the targets lie along the positive *x*-axis, positive velocity in the global coordinate system corresponds to negative closing speed. Negative velocity in the global coordinate system corresponds to positive closing speed.

Estimate the noise power after matched filtering. Create a constant noise background image for simulation purposes.

```
mfgain = matchingcoeff'*matchingcoeff;
dopgain = Np;
noisebw = fs;
noisepower = noisepow(noisebw,receiver.NoiseFigure,receiver.ReferenceTemperature);
noisepowerprc = mfgain*dopgain*noisepower;
noise = noisepowerprc*ones(size(rngdopresp));
```

Create the range and Doppler estimator objects.

```
rangeestimator = phased.RangeEstimator('NumEstimatesSource','Auto', ...
    'VarianceOutputPort',true,'NoisePowerSource','Input port', ...
    'RMSResolution',rngrms);
dopestimator = phased.DopplerEstimator('VarianceOutputPort',true, ...
    'NoisePowerSource','Input port','NumPulses',Np);
```

Locate the target indices in the range-Doppler image. Instead of using a CFAR detector, for simplicity, use the known locations and speeds of the targets to obtain the corresponding index in the range-Doppler image.

```
detidx = NaN(2,Numtgts);
tgtrng = rangeangle(tgtpos,radarpos);
```

```
tgtspd = radialspeed(tgtpos,tgtvel,radarpos,radarvel);
tgtdop = 2*speed2dop(tgtspd,c/fc);
for m = 1:numel(tgtrng)
    [~,iMin] = min(abs(rnggrid-tgtrng(m)));
    detidx(1,m) = iMin;
    [~,iMin] = min(abs(dopgrid-tgtspd(m)));
    detidx(2,m) = iMin;
end
```

Find the noise power at the detection locations.

```
ind = sub2ind(size(noise),detidx(1,:),detidx(2,:));
```

Estimate the range and range variance at the detection locations. The estimated ranges agree with the postulated ranges.

```
[rngest,rngvar] = rangeestimator(rngdopresp,rnggrid,detidx,noise(ind))
```

```
rngest = 3×1

  499.7911
  529.8380
  750.0983


rngvar = 3×1
10^-4 ×

    0.0273
    0.0276
    0.2094
```

Estimate the speed and speed variance at the detection locations. The estimated speeds agree with the predicted speeds.

```
[spdest,spdvar] = dopestimator(rngdopresp,dopgrid,detidx,noise(ind))
```

```
spdest = 3×1

   60.5241
  -19.6167
  -39.5838


spdvar = 3×1
10^-5 ×

    0.0806
    0.0816
    0.6188
```

Algorithms

Estimation Algorithm

The `phased.DopplerEstimator` System object estimates the Doppler frequency of a detection by following these steps of the Doppler estimator are

1. Input a Doppler-processed response data cube obtained from the `phased.RangeDopplerResponse` System object. The first dimension of the cube represents the fast-time or equivalent range of the returned signal samples. The second dimension represents the spatial information, such as sensors or beams. The last dimension represents the response as a function of Doppler frequency. Only this dimension is used to estimate detection Doppler frequency. All others are ignored. See “Radar Data Cube”.
2. Input the matrix of detection indices that specify the location of detections in the data cube. Each column denotes a separate detection. The row entries designate indices into the data cube. To return these detection indices as an output of the `phased.CFARDetector` or `phased.CFARDetector2D` detectors. To return these indices, set the detector `OutputFormat` property of either CFAR detector to `'Detection index'`.
3. Optionally input a row vector of cluster IDs. This vector is equal in length to the number of detections. Each element of this vector assigns an ID to a corresponding detection. To form clusters of detections, the same ID can be assigned to more than one detection. To enable this option, set the `ClusterInputPort` property to `true`.
4. When `ClusterInputPort` is `false`, the object computes Doppler frequencies for each detection. The algorithm finds the response values at the detection index and at two adjacent indices in the cube along the Doppler dimension. Then, the algorithm fits a quadratic curve to the magnitudes of the Doppler response at these three indices. The peak of the curve indicates the detection location. When detections occur at the first or last sample in the Doppler dimension, the object estimates the detection location from a two-point centroid. The centroid is formed using the location of the detection index and the sample next to the detection index.

   When the object computes Doppler frequencies for each cluster. The algorithm finds the indices of the largest response value in the cluster. Then, the algorithm fits a quadratic curve to that detection in the same way as for individual detections.
5. The object converts the fractional index values to Doppler frequency or speed by using appropriate units from the `dopgrid` input argument of the `step` method. You can obtain values for `dopgrid` using the `phased.RangeDopplerResponse` System object.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2017a

References

[1] Richards, M. *Fundamentals of Radar Signal Processing*. 2nd ed. McGraw-Hill Professional Engineering, 2014.

[2] Richards, M., J. Scheer, and W. Holm, *Principles of Modern Radar: Basic Principles*. SciTech Publishing, 2010.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

Functions
`rangeangle` | `dop2speed` | `speed2dop` | `range2bw` | `bw2range`

Objects
`phased.RangeEstimator` | `phased.RangeDopplerResponse` | `phased.CFARDetector` | `phased.CFARDetector2D`

Topics
"Radar Data Cube"

step

System object: `phased.DopplerEstimator`
Package: `phased`

Estimate target Doppler

Syntax

```
dopest = step(estimator,resp,dopgrid,detidx)
[dopest,dopvar] = step(estimator,resp,dopgrid,detidx,noisepower)
[dopest,dopvar] = step(estimator,resp,dopgrid,detidx,clusterids)
[dopest,dopvar] = step(estimator,resp,dopgrid,detidx,noisepower,clusterids)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`dopest = step(estimator,resp,dopgrid,detidx)` estimates Doppler frequencies of detections derived from the range-Doppler response data, `resp`. Doppler estimates are computed for each detection position reported in `detidx`. The `dopgrid` argument sets the units for the Doppler dimension of the response data cube.

`[dopest,dopvar] = step(estimator,resp,dopgrid,detidx,noisepower)` also specifies the noise power. This syntax applies when you set the `VarianceOutputPort` property to `true` and the `NoisePowerSource` property to `'Input port'`.

`[dopest,dopvar] = step(estimator,resp,dopgrid,detidx,clusterids)` also specifies the `clusterids` for the detections. This syntax applies when you set the `ClusterInputPort` property to `true`.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example, `[dopest,dopvar] = step(estimator,resp,dopgrid,detidx,noisepower, clusterids)`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`estimator` — Doppler estimator
`phased.DopplerEstimator` System object

Doppler estimator, specified as a `phased.DopplerEstimator` System object.

Example: `phased.DopplerEstimator`

`resp` — Doppler-processed response data cube
complex-valued P-by-1 column vector | complex-valued M-by-P matrix | complex-valued M-by-N-by-P array

Doppler-processed response data cube, specified as a complex-valued P-by-1 column vector, a complex-valued M-by-P matrix, or a complex-valued M-by-N-by-P array. M represents the number of fast-time or range samples. N is the number of spatial elements, such as sensor elements or beams. P is the number of Doppler bins.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`

`dopgrid` — Doppler grid values along Doppler dimension
real-valued P-by-1 column vector

Doppler grid values along the Doppler dimension of the `resp` argument, specified as a real-valued P-by-1 column vector. `dopgrid` defines the Doppler values corresponding to the Doppler dimension of the `resp` argument. Doppler values must be monotonically increasing and equally spaced. You can specify the grid values to be velocity or frequency. Units are in hertz or meters/sec.

Example: `[-0.3,-0.2,-0.1,0,0.1,0.2,0.3]`

Data Types: `single` | `double`

`detidx` — Detection indices
real-valued N_d-by-Q matrix

Detection indices, specified as a real-valued N_d-by-Q matrix. Q is the number of detections and N_d is the number of dimensions of the response data cube, `resp`. Each column of `detidx` contains the N_d indices of the detection in the response data cube.

To generate detection indices, you can use the `phased.CFARDetector` or `phased.CFARDetector2D` objects.

Data Types: `single` | `double`

`noisepower` — Noise power at detection locations
positive scalar | real-valued 1-by-Q row vector of positive values

Noise power at detection locations, specified as a positive scalar or real-valued 1-by-Q row vector of positive values. Q is the number of detections specified in `detidx`.

Dependencies

To enable this input argument, set the `NoisePowerSource` property to `Input port`.

Data Types: `single` | `double`

`clusterids` — Cluster IDs
real-valued 1-by-Q row vector of positive values

Cluster IDs, specified as a real-valued 1-by-*Q* row vector where *Q* is the number of detections specified in `detidx`. Each element of `clusterids` corresponds to a column in `detidx`. Detections with the same cluster ID belong to the same cluster.

Dependencies

To enable this input argument, set the `ClusterInputPort` property to `true`.

Data Types: `single` | `double`

Output Arguments

`dopest` — Doppler estimates

real-valued *K*-by-1 column vector

Doppler estimates, returned as a real-valued *K*-by-1 column vector.

- When `ClusterInputPort` is `false`, Doppler estimates are computed for each detection location in the `detidx` argument. Then *K* equals the column dimension, *Q*, of `detidx`.
- When `ClusterInputPort` is `true`, Doppler estimates are computed for each cluster ID in the `clusterids` argument. Then *K* equals the number of unique cluster IDs, *Q*.

Data Types: `single` | `double`

`dopvar` — Doppler estimation variance

positive, real-valued *K*-by-1 column vector

Doppler estimation variance, returned as a positive, real-valued *K*-by-1 column vector, where *K* is the dimension of `dopest`. Each element of `dopvar` corresponds to an element of `dopest`. The estimator variance is computed using the Ziv-Zakai bound.

Data Types: `single` | `double`

Examples

Estimate Range and Speed of Three Targets

To estimate the range and speed of three targets, create a range-Doppler map using the `phased.RangeDopplerResponse` System object™. Then use the `phased.RangeEstimator` and `phased.DopplerEstimator` System objects to estimate range and speed. The transmitter and receiver are collocated isotropic antenna elements forming a monostatic radar system.

The transmitted signal is a linear FM waveform with a pulse repetition interval (PRI) of 7.0 μs and a duty cycle of 2%. The operating frequency is 77 GHz and the sample rate is 150 MHz.

```
fs = 150e6;
c = physconst('LightSpeed');
fc = 77.0e9;
pri = 7e-6;
prf = 1/pri;
```

Set up the scenario parameters. The transmitter and receiver are stationary and located at the origin. The targets are 500, 530, and 750 meters from the radar along the *x*-axis. The targets move along the *x*-axis at speeds of –60, 20, and 40 m/s. All three targets have a nonfluctuating radar cross-section (RCS) of 10 dB. Create the target and radar platforms.

```
Numtgts = 3;
tgtpos = zeros(Numtgts);
tgtpos(1,:) = [500 530 750];
tgtvel = zeros(3,Numtgts);
tgtvel(1,:) = [-60 20 40];
tgtrcs = db2pow(10)*[1 1 1];
tgtmotion = phased.Platform(tgtpos,tgtvel);
target = phased.RadarTarget('PropagationSpeed',c,'OperatingFrequency',fc, ...
    'MeanRCS',tgtrcs);
radarpos = [0;0;0];
radarvel = [0;0;0];
radarmotion = phased.Platform(radarpos,radarvel);
```

Create the transmitter and receiver antennas.

```
txantenna = phased.IsotropicAntennaElement;
rxantenna = clone(txantenna);
```

Set up the transmitter-end signal processing. Create an upsweep linear FM signal with a bandwidth of one half the sample rate. Find the length of the PRI in samples and then estimate the rms bandwidth and range resolution.

```
bw = fs/2;
waveform = phased.LinearFMWaveform('SampleRate',fs, ...
    'PRF',prf,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',fs/2, ...
    'DurationSpecification','Duty cycle','DutyCycle',0.02);
sig = waveform();
Nr = length(sig);
bwrms = bandwidth(waveform)/sqrt(12);
rngrms = c/bwrms;
```

Set up the transmitter and radiator System object properties. The peak output power is 10 W and the transmitter gain is 36 dB.

```
peakpower = 10;
txgain = 36.0;
transmitter = phased.Transmitter( ...
    'PeakPower',peakpower, ...
    'Gain',txgain, ...
    'InUseOutputPort',true);
radiator = phased.Radiator( ...
    'Sensor',txantenna,...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc);
```

Set up the free-space channel in two-way propagation mode.

```
channel = phased.FreeSpace( ...
    'SampleRate',fs, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc, ...
    'TwoWayPropagation',true);
```

Set up the receiver-end processing. Set the receiver gain and noise figure.

```
collector = phased.Collector( ...
    'Sensor',rxantenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc);
rxgain = 42.0;
noisefig = 1;
receiver = phased.ReceiverPreamp( ...
    'SampleRate',fs, ...
    'Gain',rxgain, ...
    'NoiseFigure',noisefig);
```

Loop over the pulses to create a data cube of 128 pulses. For each step of the loop, move the target and propagate the signal. Then put the received signal into the data cube. The data cube contains the received signal per pulse. Ordinarily, a data cube has three dimensions where the last dimension corresponds to antennas or beams. Because only one sensor is used, the cube has only two dimensions.

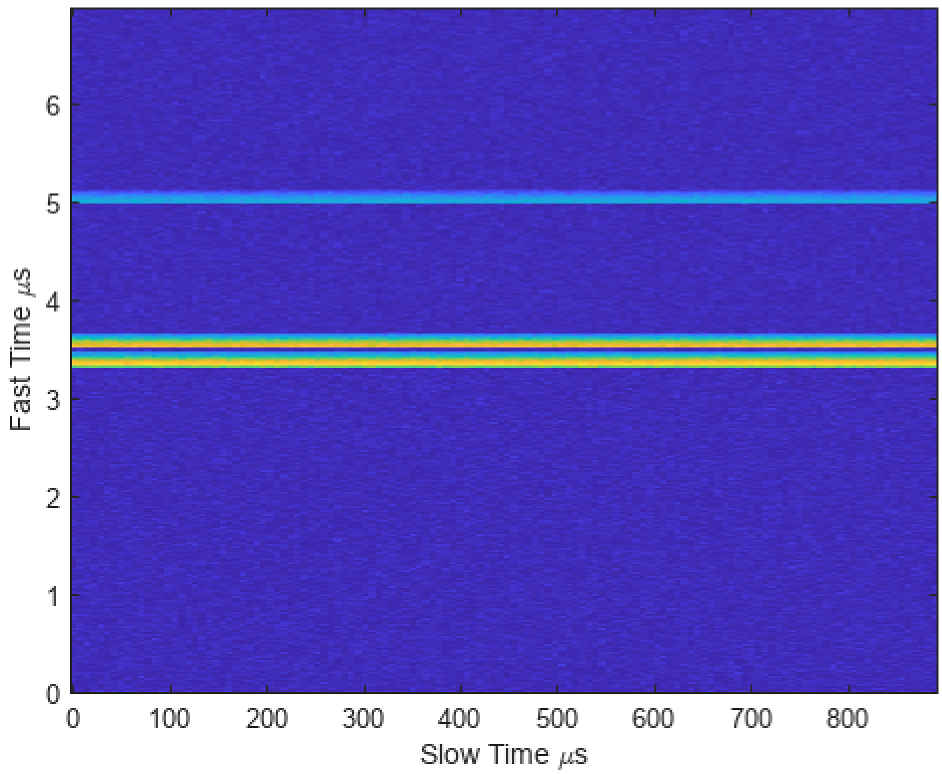
The processing steps are:

1. Move the radar and targets.
2. Transmit a waveform.
3. Propagate the waveform signal to the target.
4. Reflect the signal from the target.
5. Propagate the waveform back to the radar. Two-way propagation enables you to combine the return propagation with the outbound propagation.
6. Receive the signal at the radar.
7. Load the signal into the data cube.

```
Np = 128;
dt = pri;
cube = zeros(Nr,Np);
for n = 1:Np
    [sensorpos,sensorvel] = radarmotion(dt);
    [tgtpos,tgtvel] = tgtmotion(dt);
    [tgtrng,tgtang] = rangeangle(tgtpos,sensorpos);
    sig = waveform();
    [txsig,txstatus] = transmitter(sig);
    txsig = radiator(txsig,tgtang);
    txsig = channel(txsig,sensorpos,tgtpos,sensorvel,tgtvel);
    tgtsig = target(txsig);
    rxcol = collector(tgtsig,tgtang);
    rxsig = receiver(rxcol);
    cube(:,n) = rxsig;
end
```

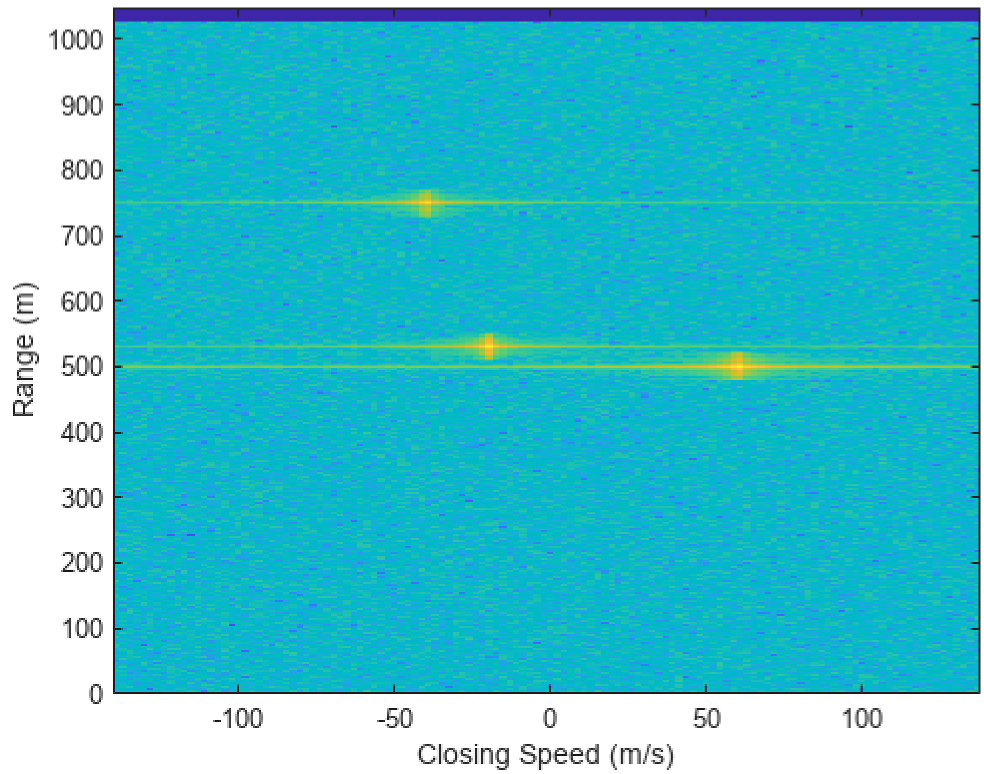
Display the data cube containing signals per pulse.

```
imagesc([0:(Np-1)]*pri*1e6,[0:(Nr-1)]/fs*1e6,abs(cube))
xlabel('Slow Time {\mu}s')
ylabel('Fast Time {\mu}s')
axis xy
```



Create and display the range-Doppler image for 128 Doppler bins. The image shows range vertically and speed horizontally. Use the linear FM waveform for match filtering. The image is here is the range-Doppler map.

```
ndop = 128;
rangedopresp = phased.RangeDopplerResponse('SampleRate',fs, ...
    'PropagationSpeed',c,'DopplerFFTLengthSource','Property', ...
    'DopplerFFTLength',ndop,'DopplerOutput','Speed', ...
    'OperatingFrequency',fc);
matchingcoeff = getMatchedFilter(waveform);
[rngdopresp,rnggrid,dopgrid] = rangedopresp(cube,matchingcoeff);
imagesc(dopgrid,rnggrid,10*log10(abs(rngdopresp)))
xlabel('Closing Speed (m/s)')
ylabel('Range (m)')
axis xy
```



Because the targets lie along the positive *x*-axis, positive velocity in the global coordinate system corresponds to negative closing speed. Negative velocity in the global coordinate system corresponds to positive closing speed.

Estimate the noise power after matched filtering. Create a constant noise background image for simulation purposes.

```
mfgain = matchingcoeff'*matchingcoeff;
dopgain = Np;
noisebw = fs;
noisepower = noisepow(noisebw,receiver.NoiseFigure,receiver.ReferenceTemperature);
noisepowerprc = mfgain*dopgain*noisepower;
noise = noisepowerprc*ones(size(rngdopresp));
```

Create the range and Doppler estimator objects.

```
rangeestimator = phased.RangeEstimator('NumEstimatesSource','Auto', ...
    'VarianceOutputPort',true,'NoisePowerSource','Input port', ...
    'RMSResolution',rngrms);
dopestimator = phased.DopplerEstimator('VarianceOutputPort',true, ...
    'NoisePowerSource','Input port','NumPulses',Np);
```

Locate the target indices in the range-Doppler image. Instead of using a CFAR detector, for simplicity, use the known locations and speeds of the targets to obtain the corresponding index in the range-Doppler image.

```
detidx = NaN(2,Numtgts);
tgtrng = rangeangle(tgtpos,radarpos);
```

```
tgtspd = radialspeed(tgtpos,tgtvel,radarpos,radarvel);
tgtdop = 2*speed2dop(tgtspd,c/fc);
for m = 1:numel(tgtrng)
    [~,iMin] = min(abs(rnggrid-tgtrng(m)));
    detidx(1,m) = iMin;
    [~,iMin] = min(abs(dopgrid-tgtspd(m)));
    detidx(2,m) = iMin;
end
```

Find the noise power at the detection locations.

```
ind = sub2ind(size(noise),detidx(1,:),detidx(2,:));
```

Estimate the range and range variance at the detection locations. The estimated ranges agree with the postulated ranges.

```
[rngest,rngvar] = rangeestimator(rngdopresp,rnggrid,detidx,noise(ind))
```

```
rngest = 3×1

  499.7911
  529.8380
  750.0983
```

```
rngvar = 3×1
10^-4 ×

    0.0273
    0.0276
    0.2094
```

Estimate the speed and speed variance at the detection locations. The estimated speeds agree with the predicted speeds.

```
[spdest,spdvar] = dopestimator(rngdopresp,dopgrid,detidx,noise(ind))
```

```
spdest = 3×1

   60.5241
  -19.6167
  -39.5838
```

```
spdvar = 3×1
10^-5 ×

    0.0806
    0.0816
    0.6188
```

Version History

Introduced in R2017a

phased.DPCACanceller

Package: `phased`

Displaced phase center array (DPCA) pulse canceller

Description

The `DPCACanceller` object implements a displaced phase center array pulse canceller for a uniform linear array (ULA).

To compute the output signal of the space time pulse canceller:

1. Define and set up your DPCA pulse canceller. See “Construction” on page 1-400.
2. Call `step` to execute the DPCA algorithm according to the properties of `phased.DPCACanceller`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.DPCACanceller` creates a displaced phase center array (DPCA) canceller System object, `H`. The object performs two-pulse DPCA processing on the input data.

`H = phased.DPCACanceller(Name,Value)` creates a DPCA object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Uniform linear array

Uniform linear array, specified as a `phased.ULA` System object.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

PRFSource

Source of pulse repetition frequency

Source of the PRF values for the STAP processor, specified as `'Property'` or `'Input port'`. When you set this property to `'Property'`, the PRF is determined by the value of the `PRF` property. When you set this property to `'Input port'`, the PRF is determined by an input argument to the `step` method at execution time.

Default: `'Property'`

PRF

Pulse repetition frequency

Pulse repetition frequency (PRF) of the received signal, specified as a positive scalar. Units are in Hertz. This property can be specified as single or double precision.

Dependencies

To enable this property, set the `PRFSource` property to `'Property'`.

Default: `1`

DirectionSource

Source of receiving mainlobe direction

Specify whether the targeting direction for the STAP processor comes from the `Direction` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `Direction` property of this object specifies the targeting direction.
`'Input port'`	An input argument in each invocation of `step` specifies the targeting direction.

Default: `'Property'`

Direction

Receiving mainlobe direction

Specify the receiving mainlobe direction of the receiving sensor array as a column vector of length 2. The direction is specified in the format of `[AzimuthAngle;ElevationAngle]` (in degrees). The azimuth angle should be between –180° and 180°. The elevation angle should be between –90° and 90°. This property applies when you set the `DirectionSource` property to `'Property'`. You can specify this argument as single or double precision.

Default: `[0; 0]`

NumPhaseShifterBits

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed. You can specify this property as single or double precision.

Default: `0`

DopplerSource

Source of targeting Doppler

Specify whether the targeting Doppler for the STAP processor comes from the `Doppler` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `Doppler` property of this object specifies the Doppler.
`'Input port'`	An input argument in each invocation of `step` specifies the Doppler.

Default: `'Property'`

Doppler

Targeting Doppler frequency (hertz)

Specify the targeting Doppler of the STAP processor as a scalar. This property applies when you set the `DopplerSource` property to `'Property'`. You can specify this property as single or double precision.

Default: `0`

WeightsOutputPort

Output processing weights

To obtain the weights used in the STAP processor, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the weights, set this property to `false`.

Default: `false`

PreDopplerOutput

Output pre-Doppler result

Set this property to `true` to output the processing result before applying the Doppler filtering. Set this property to `false` to output the processing result after the Doppler filtering.

Default: `false`

Methods

step	Perform DPCA processing on input data

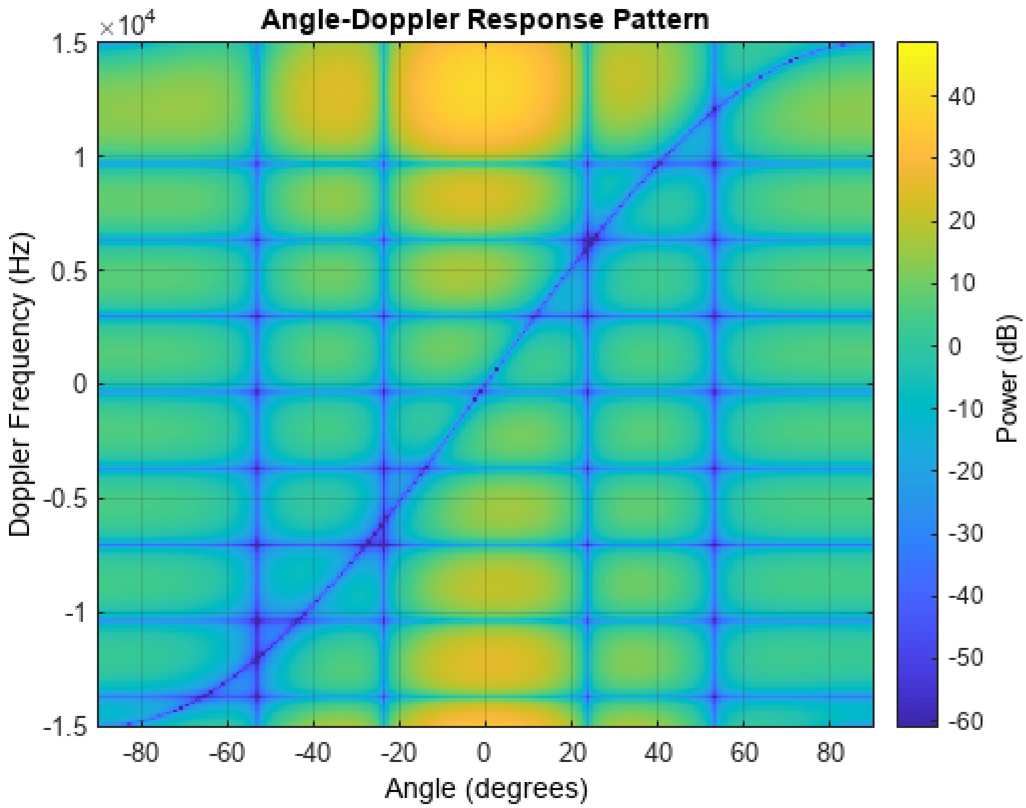
Common to All System Objects	
release	Allow System object property value changes

Examples

Process Data Cube Using DPCA

Process a data cube using a DPCA processor. The weights are calculated for the 71st cell of the collected data cube. The look direction is *(0,0)* degrees and the Doppler shift is 12.980 kHz.

```
load STAPExampleData;
Hs = phased.DPCACanceller('SensorArray',STAPEx_HArray,...
    'PRF',STAPEx_PRF,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'WeightsOutputPort',true,...
    'DirectionSource','Input port',...
    'DopplerSource','Input port');
[y,w] = step(Hs,STAPEx_ReceivePulse,71,[0;0],12.980e3);

sAngDop = phased.AngleDopplerResponse(...
    'SensorArray',Hs.SensorArray,...
    'OperatingFrequency',Hs.OperatingFrequency,...
    'PRF',Hs.PRF,...
    'PropagationSpeed',Hs.PropagationSpeed);
plotResponse(sAngDop,w)
```



Algorithms

Single Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Guerci, J. R. *Space-Time Adaptive Processing for Radar*. Boston: Artech House, 2003.

[2] Ward, J. “Space-Time Adaptive Processing for Airborne Radar Data Systems,” *Technical Report 1015*, MIT Lincoln Laboratory, December, 1994.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See Also

phased.ADPCACanceller | phased.AngleDopplerResponse | phased.STAPSMIBeamformer | uv2azel | phitheta2azel

step

System object: `phased.DPCACanceller`
Package: `phased`

Perform DPCA processing on input data

Syntax

```
Y = step(H,X,CUTIDX)
Y = step(H,X,CUTIDX,ANG)
Y = step(H,X,CUTIDX,DOP)
Y = step(H,X,CUTIDX,PRF)
[Y,W] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X,CUTIDX)` applies the DPCA pulse cancellation algorithm to the input data `X`. The algorithm calculates the processing weights according to the range cell specified by `CUTIDX`. This syntax is available when the `DirectionSource` property is `'Property'` and the `DopplerSource` property is `'Property'`. The receiving mainlobe direction is the `Direction` property value. The output `Y` contains the result of pulse cancellation either before or after Doppler filtering, depending on the `PreDopplerOutput` property value.

`Y = step(H,X,CUTIDX,ANG)` uses `ANG` as the receiving main lobe direction. This syntax is available when the `DirectionSource` property is `'Input port'` and the `DopplerSource` property is `'Property'`.

`Y = step(H,X,CUTIDX,DOP)` uses `DOP` as the targeting Doppler frequency. This syntax is available when the `DopplerSource` property is `'Input port'`.

`Y = step(H,X,CUTIDX,PRF)` uses `PRF` as the pulse repetition frequency. This syntax is available when the `PRFSource` property is `'Input port'`.

`[Y,W] = step( ___ )` also returns the processing weights, `W`. This syntax is available when the `WeightsOutputPort` property is `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Pulse canceller object.

X

Input data. X must be a 3-dimensional M-by-N-by-P numeric array whose dimensions are (range, channels, pulses). You can specify this argument as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

CUTIDX

Range cell. You can specify this argument as single or double precision.

PRF

Pulse repetition frequency specified as a positive scalar. To enable this argument, set the PRFSource property to 'Input port'. You can specify this argument as single or double precision. Units are in Hertz.

ANG

Receiving main lobe direction. ANG must be a 2-by-1 vector in the form [AzimuthAngle; ElevationAngle], in degrees. The azimuth angle must be between –180 and 180. The elevation angle must be between –90 and 90. You can specify this argument as single or double precision.

Default: Direction property of H

DOP

Targeting Doppler frequency in hertz. DOP must be a scalar. You can specify this argument as single or double precision.

Default: Doppler property of H

Output Arguments

Y

Result of applying pulse cancelling to the input data. The meaning and dimensions of Y depend on the PreDopplerOutput property of H:

- If PreDopplerOutput is true, Y contains the pre-Doppler data. Y is an M-by-(P–1) matrix. Each column in Y represents the result obtained by cancelling the two successive pulses.
- If PreDopplerOutput is false, Y contains the result of applying an FFT-based Doppler filter to the pre-Doppler data. The targeting Doppler is the Doppler property value. Y is a column vector of length M.

`W`

Processing weights the pulse canceller used to obtain the pre-Doppler data. The dimensions of `W` depend on the `PreDopplerOutput` property of `H`:

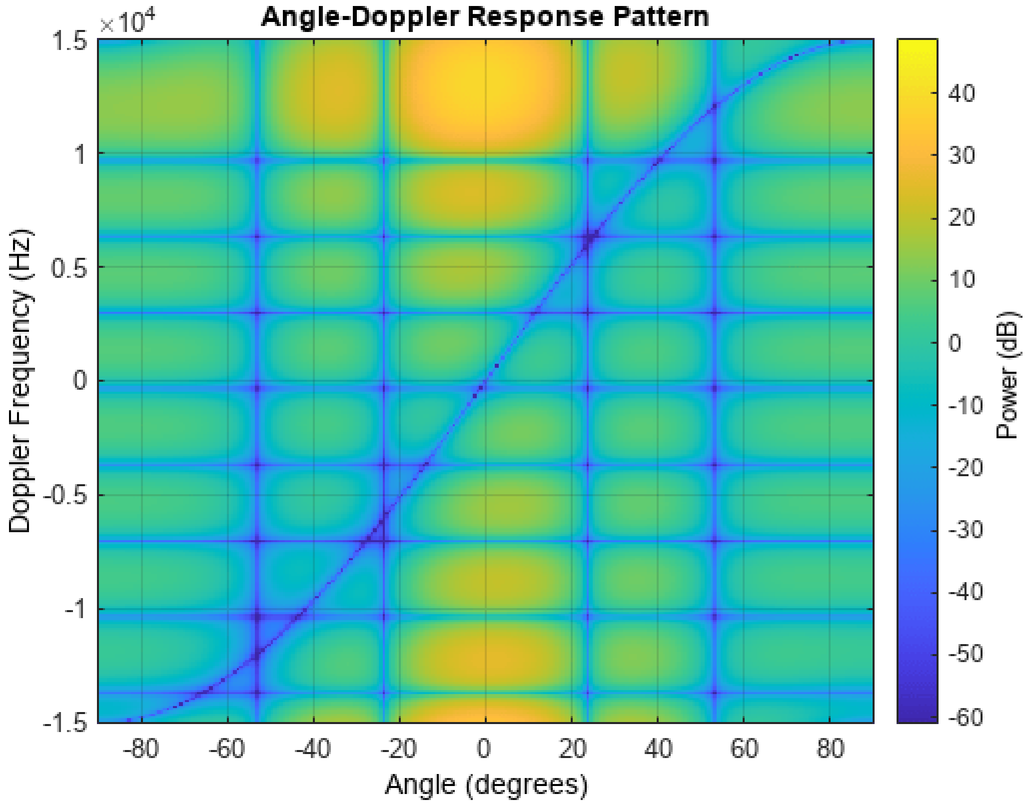
- If `PreDopplerOutput` is `true`, `W` is a 2N-by-(P-1) matrix. The columns in `W` correspond to successive pulses in `X`.
- If `PreDopplerOutput` is `false`, `W` is a column vector of length (N*P).

Examples

Process Data Cube Using DPCA

Process a data cube using a DPCA processor. The weights are calculated for the 71st cell of the collected data cube. The look direction is *(0,0)* degrees and the Doppler shift is 12.980 kHz.

```
load STAPExampleData;
Hs = phased.DPCACanceller('SensorArray',STAPEx_HArray,...
    'PRF',STAPEx_PRF,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'WeightsOutputPort',true,...
    'DirectionSource','Input port',...
    'DopplerSource','Input port');
[y,w] = step(Hs,STAPEx_ReceivePulse,71,[0;0],12.980e3);

sAngDop = phased.AngleDopplerResponse(...
    'SensorArray',Hs.SensorArray,...
    'OperatingFrequency',Hs.OperatingFrequency,...
    'PRF',Hs.PRF,...
    'PropagationSpeed',Hs.PropagationSpeed);
plotResponse(sAngDop,w)
```



See Also

`uv2azel` | `phitheta2azel`

phased.ElementDelay

Package: `phased`

Sensor array element delay estimator

Description

The `ElementDelay` object calculates the signal delay for elements in an array.

To compute the signal delay across the array elements:

1. Define and set up your element delay estimator. See "Construction" on page 1-410.
2. Call `step` to estimate the delay according to the properties of `phased.ElementDelay`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.ElementDelay` creates an element delay estimator System object, `H`. The object calculates the signal delay for elements in an array when the signal arrives the array from specified directions. By default, a 2-element uniform linear array (ULA) is used.

`H = phased.ElementDelay(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array used to calculate the delay

Specify the sensor array as a handle. The sensor array must be an array object in the `phased` package. The array cannot contain subarrays.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

Methods

step	Calculate delay for elements

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Element Delay for Uniform Linear Array

Calculate the element delay for a uniform linear array when the input is impinging on the array from 30° azimuth and 20° elevation.

```
array = phased.ULA('NumElements',4);
delay = phased.ElementDelay('SensorArray',array);
tau = delay([30;20])
```

```
tau = 4×1
10^-8 ×

    0.1175
    0.0392
   -0.0392
   -0.1175
```

Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

- This System object supports single and double precision for input data, properties, and arguments.

See Also

`phased.ArrayGain` | `phased.ArrayResponse` | `phased.SteeringVector`

step

System object: `phased.ElementDelay`
Package: `phased`

Calculate delay for elements

Syntax

`TAU = step(H,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`TAU = step(H,ANG)` returns the delay `TAU` of each element relative to the array's phase center for the signal incident directions specified by `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`H`

Element delay object.

`ANG`

Signal incident directions in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M. This argument can be single or double precision.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If `ANG` is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

Output Arguments

TAU

Delay in seconds.TAU is an N-by-M matrix, where N is the number of elements in the array. Each column of TAU contains the delays of the array elements for the corresponding direction specified in ANG. This argument can be single or double precision.

Examples

Element Delay for Uniform Linear Array

Calculate the element delay for a uniform linear array when the input is impinging on the array from 30° azimuth and 20° elevation.

```
array = phased.ULA('NumElements',4);
delay = phased.ElementDelay('SensorArray',array);
tau = delay([30;20])
```

```
tau = 4×1
10^-8 ×

    0.1175
    0.0392
   -0.0392
   -0.1175
```

See Also

uv2azel | phitheta2azel

phased.ESPRITEstimator

Package: `phased`

ESPRIT direction of arrival (DOA) estimator for ULA

Description

The `phased.ESPRITEstimator` System object estimate the direction of arrival of signals parameters via rotational invariance (ESPRIT) direction of arrival estimate.

To estimate the direction of arrival (DOA):

1. Define and set up your DOA estimator. See "Construction" on page 1-415.
2. Call `step` to estimate the DOA according to the properties of `phased.ESPRITEstimator`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.ESPRITEstimator` creates an ESPRIT DOA estimator System object, `H`. The object estimates the signal's direction-of-arrival (DOA) using the ESPRIT algorithm with a uniform linear array (ULA).

`H = phased.ESPRITEstimator(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be a `phased.ULA` object.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

ForwardBackwardAveraging

Perform forward-backward averaging

Set this property to `true` to use forward-backward averaging to estimate the covariance matrix for sensor arrays with conjugate symmetric array manifold.

Default: `false`

SpatialSmoothing

Spatial smoothing

Specify the number of averaging used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each additional smoothing handles one extra coherent source, but reduces the effective number of element by 1. The maximum value of this property is M–2, where M is the number of sensors. You can specify this property as single or double precision.

Default: `0`, indicating no spatial smoothing

NumSignalsSource

Source of number of signals

Specify the source of the number of signals as one of `'Auto'` or `'Property'`. If you set this property to `'Auto'`, the number of signals is estimated by the method specified by the `NumSignalsMethod` property.

Default: `'Auto'`

NumSignalsMethod

Method to estimate number of signals

Specify the method to estimate the number of signals as one of `'AIC'` or `'MDL'`. The `'AIC'` uses the Akaike Information Criterion and the `'MDL'` uses Minimum Description Length criterion. This property applies when you set the `NumSignalsSource` property to `'Auto'`.

Default: `'AIC'`

NumSignals

Number of signals

Specify the number of signals as a positive integer scalar. This property applies when you set the `NumSignalsSource` property to `'Property'`. The number of signals, N_{sig}, must be smaller than the number of elements, N_{sub}, in the subarray derived from the array specified in the `SensorArray` property. See “ESPRIT Subarrays” on page 1-418. You can specify this property as single or double precision.

Default: 1

`Method`

Type of least squares method

Specify the least squares method used for ESPRIT as one of `'TLS'` or `'LS'`. `'TLS'` refers to total least squares and `'LS'`refers to least squares.

Default: `'TLS'`

`RowWeighting`

Row weighting factor

Specify the row weighting factor for signal subspace eigenvectors as a positive integer scalar. This property controls the weights applied to the selection matrices. In most cases the higher value the better. However, it can never be greater than *(Nsub – 1)/2* where *Nsub* is the number of elements in the subarray derived from the array specified in the `SensorArray` property. See “ESPRIT Subarrays” on page 1-418. You can specify this property as single or double precision.

Default: 1

Methods

step	Perform DOA estimation

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Estimate DOAs of Two Signals

Estimate the directions-of-arrival (DOA) of two signals received by a standard 10-element ULA with element spacing 1 m. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 45° in azimuth and 60° in elevation.

Create the signals.

```
fs = 8.0e3;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
array.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
```

Create the plane waves and add noise.

```
x = collectPlaneWave(array,[x1 x2],[10 20;45 60]',fc);
noise = 0.1/sqrt(2)*(randn(size(x)) + 1i*randn(size(x)));
```

Estimate the arrival angles.

```
estimator = phased.ESPRITEstimator('SensorArray',array,...
    'OperatingFrequency',fc);
doas = estimator(x + noise);
az = broadside2az(sort(doas),[20 60])
```

```
az = 1×2

   10.0000   45.0126
```

Algorithms

ESPRIT Subarrays

The ESPRIT algorithm, as implemented in the `phased.ESPRITEstimator` System object, reorganizes the ULA elements into two overlapping subarrays. For an original N-element array, the first subarray consist of elements $1,...,N - 1$ of the original array. The second subarray consist of elements $2, ... ,N$ of the original array. There are $N_{sub} = N - 1$ elements in each subarray.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`broadside2az`

step

System object: `phased.ESPRITEstimator`
Package: `phased`

Perform DOA estimation

Syntax

`ANG = step(H,X)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`ANG = step(H,X)` estimates the DOAs from `X` using the DOA estimator, `H`. `X` is a matrix whose columns correspond to channels. `ANG` is a row vector of the estimated broadside angles (in degrees). You can specify this argument as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Estimate DOAs of Two Signals

Estimate the directions-of-arrival (DOA) of two signals received by a standard 10-element ULA with element spacing 1 m. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 45° in azimuth and 60° in elevation.

Create the signals.

```
fs = 8.0e3;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
```

```
array.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
```

Create the plane waves and add noise.

```
x = collectPlaneWave(array,[x1 x2],[10 20;45 60]',fc);
noise = 0.1/sqrt(2)*(randn(size(x)) + 1i*randn(size(x)));
```

Estimate the arrival angles.

```
estimator = phased.ESPRITEstimator('SensorArray',array,...
    'OperatingFrequency',fc);
doas = estimator(x + noise);
az = broadside2az(sort(doas),[20 60])
```

```
az = 1×2

   10.0000   45.0126
```

phased.FMCWWaveform

Package: `phased`

FMCW waveform

Description

The `FMCWWaveform` object creates an FMCW (frequency modulated continuous wave) waveform.

To obtain waveform samples:

1. Define and set up your FMCW waveform. See "Construction" on page 1-422.
2. Call `step` to generate the FMCW waveform samples according to the properties of `phased.FMCWWaveform`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

Construction

`H = phased.FMCWWaveform` creates an FMCW waveform System object, `H`. The object generates samples of an FMCW waveform.

`H = phased.FMCWWaveform(Name,Value)` creates an FMCW waveform object, `H`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name on page 1-422, and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,…,NameN,ValueN`.

Properties

`SampleRate`

Sample rate

Specify the same rate, in hertz, as a positive scalar. The default value of this property corresponds to 1 MHz.

The quantity `(SampleRate .* SweepTime)` is a scalar or vector that must contain only integers.

Default: `1e6`

`SweepTime`

Duration of each linear FM sweep

Specify the duration of the upsweep or downsweep, in seconds, as a row vector of positive, real numbers. The default value corresponds to 100 μs.

If `SweepDirection` is `'Triangle'`, the sweep time is half the sweep period because each period consists of an upsweep and a downsweep. If `SweepDirection` is `'Up'` or `'Down'`, the sweep time equals the sweep period.

The quantity `(SampleRate .* SweepTime)` is a scalar or vector that must contain only integers.

To implement a varying sweep time, specify `SweepTime` as a nonscalar row vector. The waveform uses successive entries of the vector as the sweep time for successive periods of the waveform. If the last element of the vector is reached, the process continues cyclically with the first entry of the vector.

If `SweepTime` and `SweepBandwidth` are both nonscalar, they must have the same length.

Default: `1e-4`

`SweepBandwidth`

FM sweep bandwidth

Specify the bandwidth of the linear FM sweeping, in hertz, as a row vector of positive, real numbers. The default value corresponds to 100 kHz.

To implement a varying bandwidth, specify `SweepBandwidth` as a nonscalar row vector. The waveform uses successive entries of the vector as the sweep bandwidth for successive periods of the waveform. If the last element of the `SweepBandwidth` vector is reached, the process continues cyclically with the first entry of the vector.

If `SweepTime` and `SweepBandwidth` are both nonscalar, they must have the same length.

Default: `1e5`

`SweepDirection`

FM sweep direction

Specify the direction of the linear FM sweep as one of `'Up'` | `'Down'` | `'Triangle'`.

Default: `'Up'`

`SweepInterval`

Location of FM sweep interval

If you set this property value to `'Positive'`, the waveform sweeps in the interval between 0 and *B*, where *B* is the `SweepBandwidth` property value. If you set this property value to `'Symmetric'`, the waveform sweeps in the interval between –*B*/2 and *B*/2.

Default: `'Positive'`

`OutputFormat`

Output signal format

Specify the format of the output signal as one of `'Sweeps'` or `'Samples'`. When you set the `OutputFormat` property to `'Sweeps'`, the output of the `step` method is in the form of multiple sweeps. In this case, the number of sweeps is the value of the `NumSweeps` property. If the `SweepDirection` property is `'Triangle'`, each sweep is half a period.

When you set the `OutputFormat` property to `'Samples'`, the output of the `step` method is in the form of multiple samples. In this case, the number of samples is the value of the `NumSamples` property.

Default: `'Sweeps'`

NumSamples

Number of samples in output

Specify the number of samples in the output of the `step` method as a positive integer. This property applies only when you set the `OutputFormat` property to `'Samples'`.

Default: `100`

NumSweeps

Number of sweeps in output

Specify the number of sweeps in the output of the `step` method as a positive integer. This property applies only when you set the `OutputFormat` property to `'Sweeps'`.

Default: `1`

Methods

plot	Plot FMCW waveform
reset	Reset states of FMCW waveform object
step	Samples of FMCW waveform

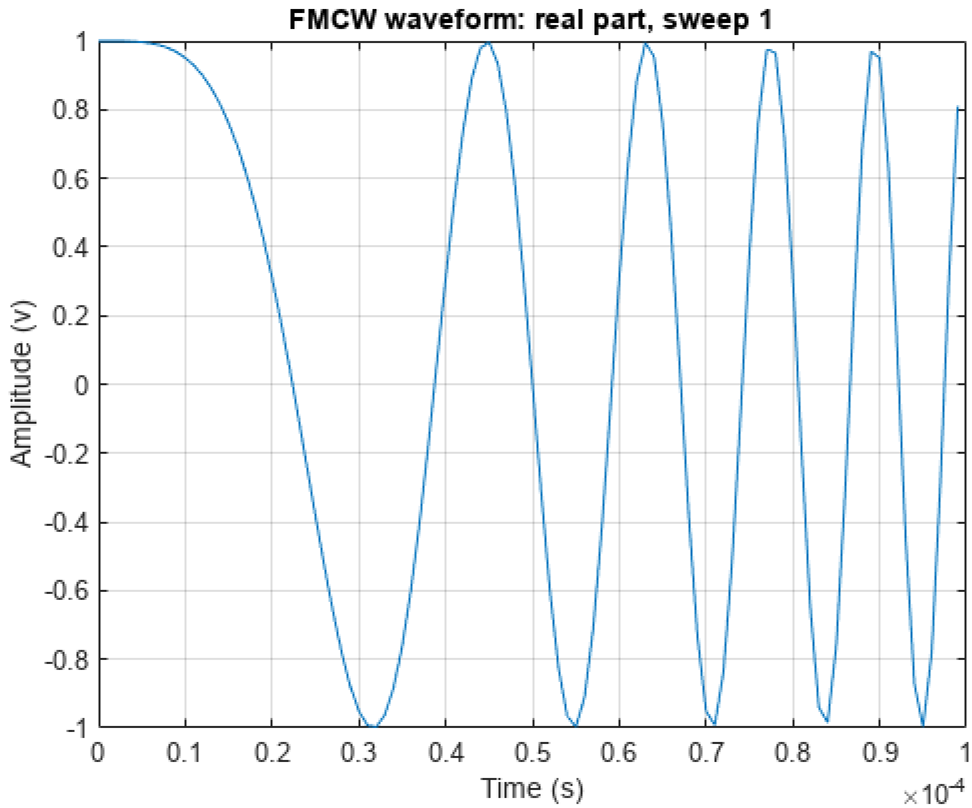
Common to All System Objects	
`release`	Allow System object property value changes

Examples

Plot FMCW Waveform

Create and plot an upsweep FMCW waveform.

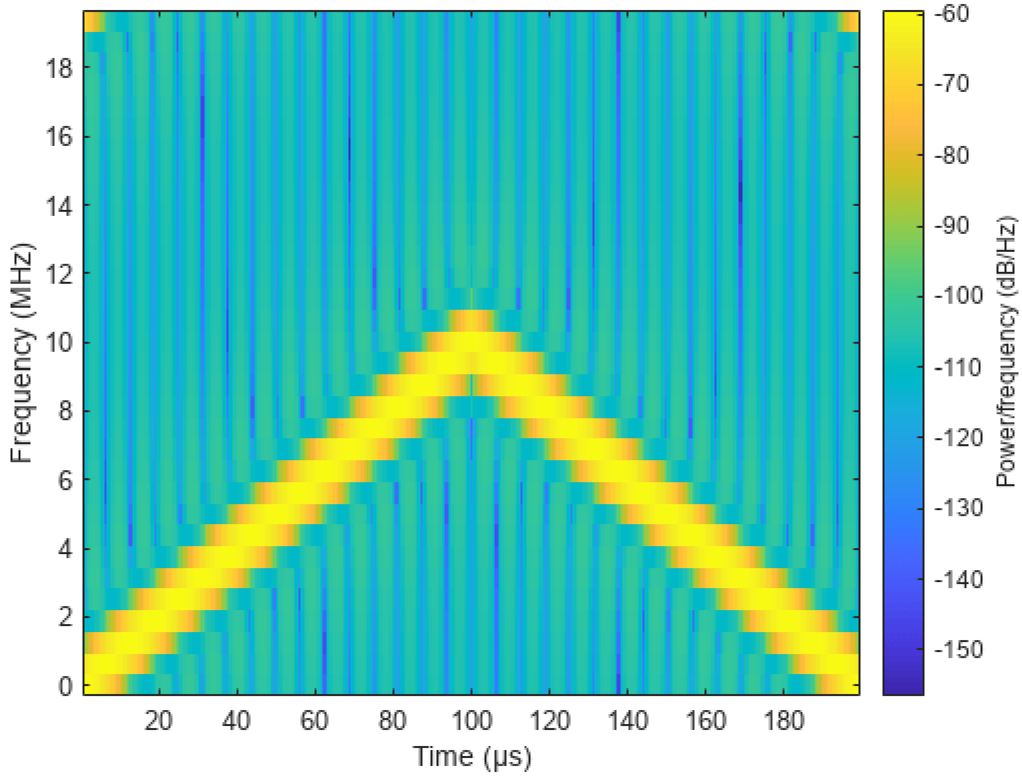
```
waveform = phased.FMCWWaveform('SweepBandwidth',100.0e3,...
    'OutputFormat','Sweeps','NumSweeps',2);
plot(waveform)
```



Spectrogram of Triangle Sweep FMCW Waveform

Generate samples of a triangle sweep FMCW Waveform. Then, plot the spectrogram of the sweep. The sweep has a 10 MHz bandwidth.

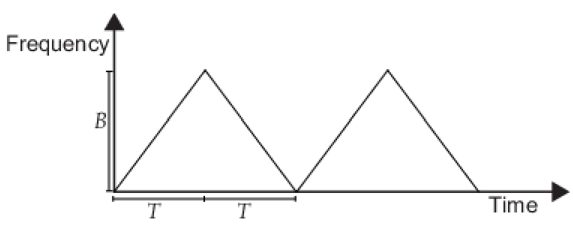
```
sFMCW = phased.FMCWWaveform('SweepBandwidth',10.0e6,...
   'SampleRate',20.0e6,'SweepDirection','Triangle',...
   'NumSweeps',2);
sig = step(sFMCW);
windowlength = 32;
noverlap = 16;
nfft = 32;
spectrogram(sig,windowlength,noverlap,nfft,sFMCW.SampleRate,'yaxis')
```



More About

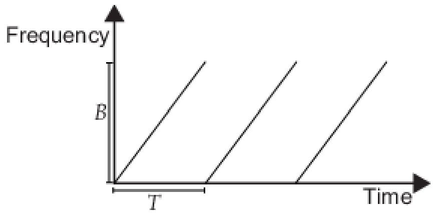
Triangle Sweep

In each period of a triangle sweep, the waveform sweeps up with a slope of *B*/*T* and then down with a slope of –*B*/*T*. *B* is the sweep bandwidth, and *T* is the sweep time. The sweep period is 2*T*.



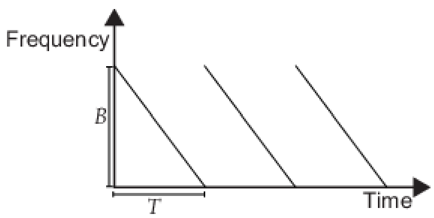
Upsweep

In each period of an upsweep, the waveform sweeps with a slope of *B*/*T*. *B* is the sweep bandwidth, and *T* is the sweep time.



Downsweep

In each period of a downsweep, the waveform sweeps with a slope of –*B*/*T*. *B* is the sweep bandwidth, and *T* is the sweep time.



Version History

Introduced in R2012b

References

[1] Issakov, Vadim. *Microwave Circuits for 24 GHz Automotive Radar in Silicon-based Technologies*. Berlin: Springer, 2010.

[2] Skolnik, M.I. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `plot` method is not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`range2time` | `time2range` | `range2bw` | `phased.LinearFMWaveform`

Topics

"Automotive Adaptive Cruise Control Using FMCW Technology" (Radar Toolbox)

plot

System object: `phased.FMCWWaveform`
Package: `phased`

Plot FMCW waveform

Syntax

```
plot(Hwav)
plot(Hwav,Name,Value)
plot(Hwav,Name,Value,LineSpec)
h = plot( ___ )
```

Description

`plot(Hwav)` plots the real part of the waveform specified by `Hwav`.

`plot(Hwav,Name,Value)` plots the waveform with additional options specified by one or more `Name,Value` pair arguments.

`plot(Hwav,Name,Value,LineSpec)` specifies the same line color, line style, or marker options as are available in the MATLAB `plot` function.

`h = plot( ___ )` returns the line handle in the figure.

Input Arguments

`Hwav`

Waveform object. This variable must be a scalar that represents a single waveform object.

`LineSpec`

Character vector to specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `PlotType` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Default: `'b'`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PlotType`

Specifies whether the function plots the real part, imaginary part, or both parts of the waveform. Valid values are `'real'`, `'imag'`, and `'complex'`.

Default: `'real'`

SweepIdx

Index of the sweep to plot. This value must be a positive integer scalar.

Default: `1`

Output Arguments

h

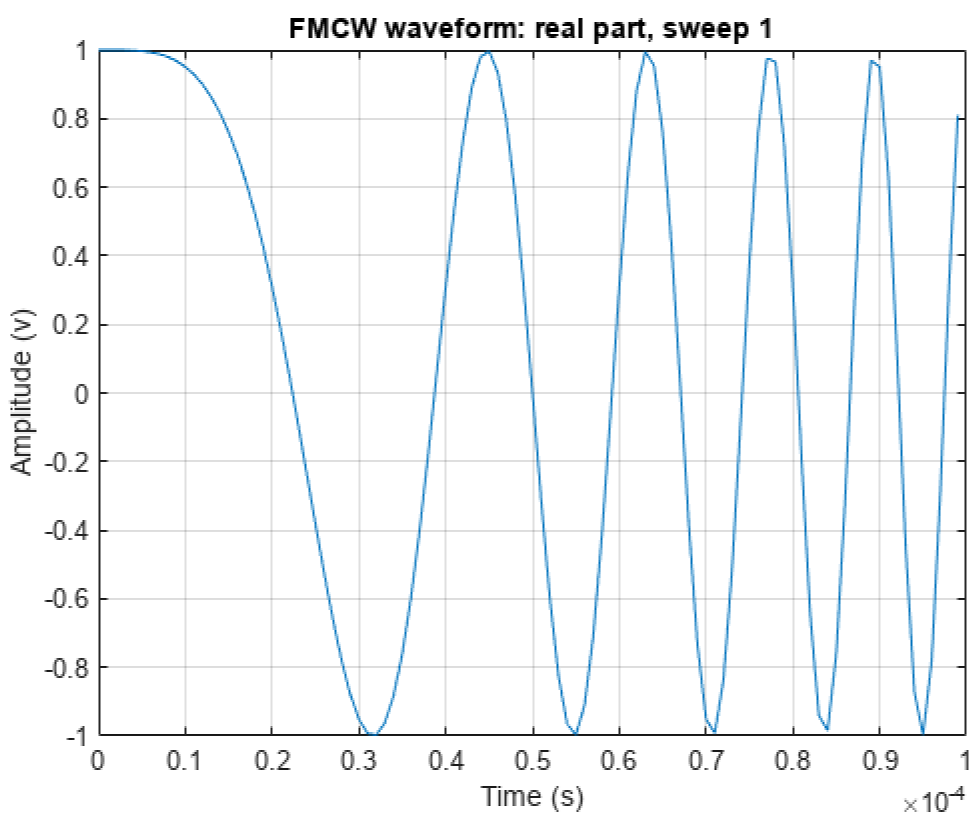
Handle to the line or lines in the figure. For a `PlotType` value of `'complex'`, `h` is a column vector. The first and second elements of this vector are the handles to the lines in the real and imaginary subplots, respectively.

Examples

Plot FMCW Waveform

Create and plot an upsweep FMCW waveform.

```
waveform = phased.FMCWWaveform('SweepBandwidth',100.0e3,...
    'OutputFormat','Sweeps','NumSweeps',2);
plot(waveform)
```



reset

System object: `phased.FMCWWaveform`
Package: `phased`

Reset states of FMCW waveform object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `FMCWWaveform` object, `H`. Afterward, the next call to `step` restarts the sweep of the waveform.

step

System object: `phased.FMCWWaveform`
Package: `phased`

Samples of FMCW waveform

Syntax

`Y = step(H)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

`Y = step(H)` returns samples of the FMCW waveform in a column vector, `Y`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

FMCW waveform object.

Output Arguments

Y

Column vector containing the waveform samples.

If `H.OutputFormat` is `'Samples'`, `Y` consists of `H.NumSamples` samples.

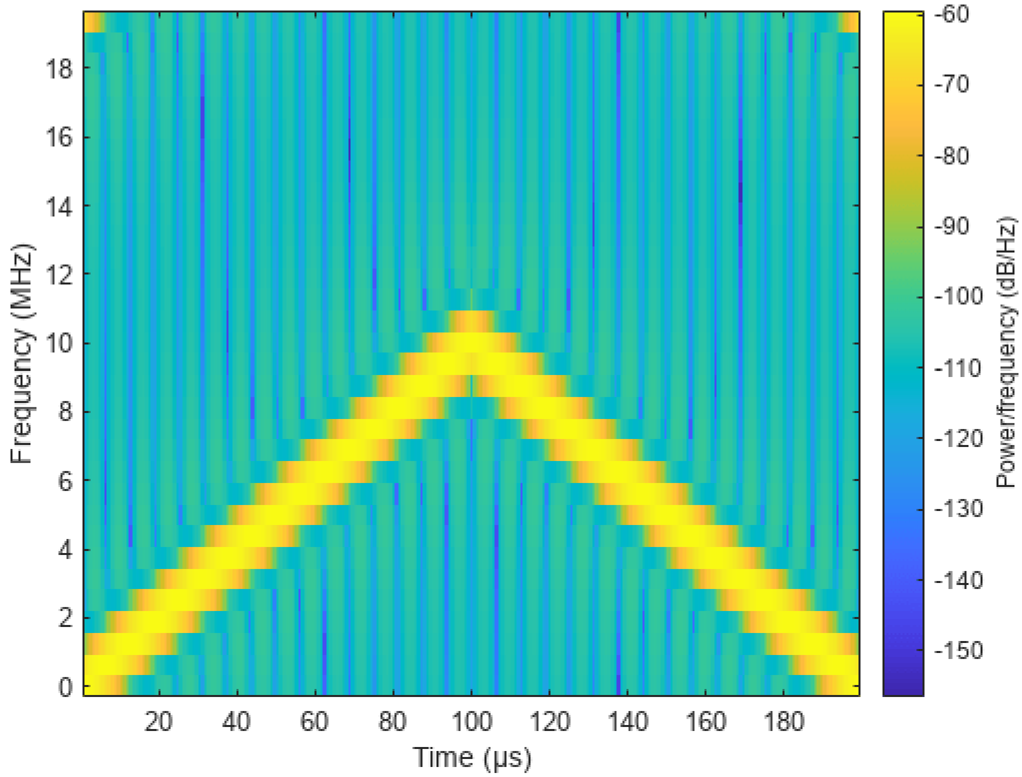
If `H.OutputFormat` is `'Sweeps'`, `Y` consists of `H.NumSweeps` sweeps. Also, if `H.SweepDirection` is `'Triangle'`, each sweep is half a period.

Examples

Spectrogram of Triangle Sweep FMCW Waveform

Generate samples of a triangle sweep FMCW Waveform. Then, plot the spectrogram of the sweep. The sweep has a 10 MHz bandwidth.

```
sFMCW = phased.FMCWWaveform('SweepBandwidth',10.0e6,...
   'SampleRate',20.0e6,'SweepDirection','Triangle',...
   'NumSweeps',2);
sig = step(sFMCW);
windowlength = 32;
noverlap = 16;
nfft = 32;
spectrogram(sig,windowlength,noverlap,nfft,sFMCW.SampleRate,'yaxis')
```



phased.FreeSpace

Package: `phased`

Free space environment

Description

The `phased.FreeSpace` System object models narrowband signal propagation from one point to another in a free-space environment. The object applies range-dependent time delay, gain and phase shift to the input signal. The object accounts for Doppler shift when either the source or destination is moving. A free-space environment is a boundaryless medium with a speed of signal propagation independent of position and direction. The signal propagates along a straight line from source to destination. For example, you can use this object to model the propagation of a signal from a radar to a target and back to the radar.

For non-polarized signals, the `FreeSpace` System object lets you propagate signals from a single point to multiple points or from multiple points to a single point. Multiple-point to multiple-point propagation is not supported.

To compute the propagated signal in free space:

1. Define and set up your free space environment. See "Construction" on page 1-434.
2. Call `step` to propagate the signal through a free space environment according to the properties of `phased.FreeSpace`. The behavior of `step` is specific to each object in the toolbox.

When propagating a round trip signal in free-space, you can either use one `FreeSpace` System object to compute the two-way propagation delay or two separate `FreeSpace` System objects to compute one-way propagation delays in each direction. Due to filter distortion, the total round trip delay when you employ two-way propagation can differ from the delay when you use two one-way `phased.FreeSpace` System objects. It is more accurate to use a single two-way `phased.FreeSpace` System object. This option is set by the `TwoWayPropagation` property.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.FreeSpace` creates a free space environment System object, `H`.

`H = phased.FreeSpace(Name,Value)` creates a free space environment object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`PropagationSpeed`

Signal propagation speed

Specify signal wave propagation speed in free space as a real positive scalar. Units are meters per second.

Default: Speed of light

OperatingFrequency

Signal carrier frequency

A scalar containing the carrier frequency of the narrowband signal. Units are hertz.

Default: `3e8`

TwoWayPropagation

Perform two-way propagation

Set this property to `true` to perform round-trip propagation between the origin and destination that you specify in the `step` command. Set this property to `false` to perform one-way propagation from the origin to the destination.

Default: `false`

SampleRate

Sample rate

A scalar containing the sample rate. Units of sample rate are hertz. The algorithm uses this value to determine the propagation delay in number of samples.

Default: `1e6`

MaximumDistanceSource

Source of maximum distance value

Source of maximum distance value, specified as `'Auto'` or `'Property'`. This choice selects how the maximum one-way propagation distance is determined. The maximum one-way propagation distance is used to allocate sufficient memory for delay computation. When you set this property to `'Auto`, the System object automatically allocates memory. When you set this property to `'Property'`, you specify the maximum one-way propagation distance using the value of the `MaximumDistance` property.

Default: `'Auto'`

MaximumDistance

Maximum one-way propagation distance

Maximum one-way propagation distance, specified as a real-valued positive scalar. Units are meters. This property applies when you set the `MaximumDistanceSource` property to `'Property'`. Any signal that propagates more than the maximum one-way distance is ignored. The maximum distance should be greater than or equal to the largest position-to-position distance.

Default: `10000`

`MaximumNumInputSamplesSource`

Source of maximum number of samples.

The source of the maximum number of samples in the input signal, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the propagation model automatically allocates enough memory to buffer the first input signal. When you set this property to `'Property'`, specify the maximum number of samples in the input signal using the `MaximumNumInputSamples` property. Any input signal longer than that value is truncated.

This property applies when you set the `MaximumDistanceSource` property to `'Property'`.

To use this object with variable-size input signals in a MATLAB Function Block in Simulink®, set the `MaximumNumInputSamplesSource` property to `'Property'` and set a value for the `MaximumNumInputSamples` property.

Default: `'Auto'`

`MaximumNumInputSamples`

Maximum number of input signal samples.

Maximum number of samples in the input signal, specified as a positive integer. This property limits the size of the input signal. Any input signal longer than this value is truncated. The input signal is the first argument to the `step` method. The number of samples is the number of rows in the input.

This property applies when you set the `MaximumNumInputSamplesSource` property to `'Property'`.

Default: `100`

Methods

reset	Reset internal states of propagation channel
step	Propagate signal from one location to another

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Signal Propagation from Stationary Radar to Stationary Target

Calculate the amplitude of a signal propagating in free-space from a radar at (40000,0,0) to a target at (300,200,50). Assume both the radar and the target are stationary. The sample rate is 8000 Hz while the operating frequency of the radar is 300 MHz. Transmit five samples of a unit amplitude signal. The signal propagation speed takes the default value of the speed of light. Examine the amplitude of the signal at the target.

```
fs = 8e3;
fop = 3e8;
```

```
freesp = phased.FreeSpace(SampleRate=fs, ...
    OperatingFrequency=fop);
pos1 = [40000;0;0];
pos2 = [300;200;50];
vel1 = [0;0;0];
vel2 = [0;0;0];
```

Create the transmitted signal.

```
x = ones(5,1);
```

Find the received signal at the target.

```
y = freesp(x,pos1,pos2,vel1,vel2);
disp(y)
```

```
   1.0e-05 *

   0.0000 + 0.0000i
   0.1870 - 0.0229i
   0.1988 - 0.0243i
   0.1988 - 0.0243i
   0.1988 - 0.0243i
```

The first sample is zero because the signal has not yet reached the target.

Manually compute the loss using the formula

$$L = (4\pi R/\lambda)^2$$

```
R = sqrt((pos1-pos2)'*(pos1-pos2));
lambda = physconst('Lightspeed')/fop;
L = (4*pi*R/lambda)^2
```

```
L = 2.4924e+11
```

Because the transmitted amplitude is unity, the magnitude-squared value of the signal at the target for the third sample equals the inverse of the loss.

```
disp(1/abs(y(3))^2)
```

```
   2.4924e+11
```

Signal Propagation from Moving Radar to Moving Target

Calculate the result of propagating a signal in free space from a radar at (1000,0,0) to a target at (300,200,50). Assume the radar moves at 10 m/s along the *x*-axis, while the target moves at 15 m/s along the *y*-axis. The sample rate is 8000 Hz while the operating frequency of the radar is 300 MHz. The signal propagation speed takes the default value of the speed of light. Transmit five samples of a unit amplitude signal and examine the amplitude of the signal at the target.

```
fs = 8000;
fop = 3e8;
freesp = phased.FreeSpace(SampleRate=fs, ...
    OperatingFrequency=fop);
```

```
pos1 = [1000;0;0];
pos2 = [300;200;50];
vel1 = [10;0;0];
vel2 = [0;15;0];
y = freesp(ones(5,1),pos1,pos2,vel1,vel2);
disp(y)
```

```
   1.0e-03 *

   0.0126 - 0.1061i
   0.0117 - 0.1083i
   0.0105 - 0.1085i
   0.0094 - 0.1086i
   0.0082 - 0.1087i
```

Because the transmitted amplitude is unity, the square of the signal at the target equals the inverse of the loss.

```
disp(1/abs(y(2))^2)
```

```
   8.4206e+07
```

More About

Freespace Time Delay and Path Loss

When the origin and destination are stationary relative to each other, you can write the output signal of a free-space channel as $Y(t) = x(t-\tau)/L_{fsp}$. The quantity τ is the signal delay and L_{fsp} is the free-space path loss. The delay τ is given by R/c, where R is the propagation distance and c is the propagation speed. The free-space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far field of the transmitting element or array. In the near field, the free-space path loss formula is not valid and can result in a loss smaller than one, equivalent to a signal gain. Therefore, the loss is set to unity for range values, $R \le \lambda/4\pi$.

When the origin and destination have relative motion, the processing also introduces a Doppler frequency shift. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The quantity v is the relative speed of the destination with respect to the origin.

For more details on free space channel propagation, see [2].

Version History

Introduced in R2011a

References

[1] Proakis, J. *Digital Communications*. New York: McGraw-Hill, 2001.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Requires dynamic memory allocation. See “Limitations for System Objects that Require Dynamic Memory Allocation”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`fspl` | `phased.RadarTarget`

reset

System object: `phased.FreeSpace`
Package: `phased`

Reset internal states of propagation channel

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `FreeSpace` object, `H`.

step

System object: `phased.FreeSpace`
Package: `phased`

Propagate signal from one location to another

Syntax

`Y = step(SFS,F,origin_pos,dest_pos,origin_vel,dest_vel)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(SFS,F,origin_pos,dest_pos,origin_vel,dest_vel)` returns the resulting signal `Y` when the narrowband signal `F` propagates in free space from the position or positions specified in `origin_pos` to the position or positions specified in `dest_pos`. For non-polarized signals, either the `origin_pos` or `dest_pos` arguments can specify more than one point. Using both arguments to specify multiple points is not allowed. The velocity of the signal origin is specified in `origin_vel` and the velocity of the signal destination is specified in `dest_vel`. The dimensions of `origin_vel` and `dest_vel` must agree with the dimensions of `origin_pos` and `dest_pos`, respectively.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

SFS — Free-space propagator

System object

Free-space propagator, specified as a System object.

F — Narrowband signal

M-element complex-valued column vector, *M*-by-*N* complex-valued matrix or structure containing complex-valued fields.

Narrowband signal, specified as an *M*-element complex-valued column vector, *M*-by-*N* complex-valued matrix or structure containing complex-valued fields.

Polarization	Signal structure
Not enabled	The signal `X` can be a complex-valued 1-by-*M* column vector or complex-valued *M*-by-*N* matrix. The quantity *M* is the number of sample values of the signal and *N* is the number of signals to propagate. When you specify *N* signals, you need to specify *N* signal origins or *N* signal destinations. The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
Enabled	The signal `X` is a MATLAB `struct` containing three matrix fields:`X.X`, `X.Y`, and `X.Z` representing the *x*, *y*, and *z* components of the polarized signals. The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

`origin_pos` — Signal origin
vector | matrix

Origin of the signal or signals, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. Position units are meters. The quantity *N* is the number of signals arriving from *N* signal origins and matches the dimension specified in the signal `X`. If `origin_pos` is a column vector, it takes the form `[x; y; z]`. If `origin_pos` is a matrix, each column specifies a different signal origin and has the form `[x; y; z]`. `origin_pos` and `dest_pos` cannot both be specified as matrices — at least one must be a 3-by-1 column vector.

`dest_pos` — Signal destination
vector | matrix

Destination of the signal or signals, specified as a 3-by-1 column vector or 3-by-*N* matrix. Position units are meters. The quantity *N* is the number of signals arriving at *N* signal destinations and matches the dimension specified in the signal `X`. If `dest_pos` is a column vector, it takes the form `[x; y; z]`. If `dest_pos` is a matrix, each column specifies a different destination and has the form `[x; y; z]`. `dest_pos` and `origin_pos` cannot both be specified as matrices — at least one must be a 3-by-1 column vector.

`origin_vel` — Velocity of signal origin
vector | matrix

Velocity of signal origin, specified as a 3-by-1 column vector or 3-by-*N* matrix. Velocity units are meters/second. The dimensions of `origin_vel` must match the dimensions of `origin_pos`. If `origin_vel` is a column vector, it takes the form `[Vx; Vy; Vz]`. If `origin_vel` is a 3–by-*N* matrix, each column specifies a different origin velocity and has the form `[Vx; Vy; Vz]`.

dest_vel — Velocity of signal destinations
vector | matrix

Velocity of signal destinations, specified as a 3-by-1 column vector or 3–by-*N* matrix. Velocity units are meters/second. The dimensions of `dest_vel` must match the dimensions of `dest_pos`. If `dest_vel` is a column vector, it takes the form `[Vx; Vy; Vz]`. If `dest_vel` is a 3–by-*N* matrix, each column specifies a different destination velocity and has the form `[Vx; Vy; Vz]`.

Output Arguments

Y

Propagated signal, returned as a *M*-element complex-valued column vector, *M*-by-*N* complex-valued matrix or MATLAB structure containing complex-valued fields.

If X is a column vector or matrix, Y is also a column vector or matrix with the same dimensions.

If X is a `struct`, Y is also a `struct` with the same fields. Each field in Y contains the resulting signal of the corresponding field in X.

The output Y contains signal samples arriving at the signal destination within the current time frame. The current time frame is defined as the time spanned by the current input. Whenever it takes longer than the current time frame for the signal to propagate from the origin to the destination, the output contains no contribution from the input of the current time frame.

Examples

Signal Propagation from Stationary Radar to Stationary Target

Calculate the amplitude of a signal propagating in free-space from a radar at (40000,0,0) to a target at (300,200,50). Assume both the radar and the target are stationary. The sample rate is 8000 Hz while the operating frequency of the radar is 300 MHz. Transmit five samples of a unit amplitude signal. The signal propagation speed takes the default value of the speed of light. Examine the amplitude of the signal at the target.

```
fs = 8e3;
fop = 3e8;
freesp = phased.FreeSpace(SampleRate=fs, ...
    OperatingFrequency=fop);
pos1 = [40000;0;0];
pos2 = [300;200;50];
vel1 = [0;0;0];
vel2 = [0;0;0];
```

Create the transmitted signal.

```
x = ones(5,1);
```

Find the received signal at the target.

```
y = freesp(x,pos1,pos2,vel1,vel2);
disp(y)
```

```
   1.0e-05 *
```

```
   0.0000 + 0.0000i
   0.1870 - 0.0229i
   0.1988 - 0.0243i
   0.1988 - 0.0243i
   0.1988 - 0.0243i
```

The first sample is zero because the signal has not yet reached the target.

Manually compute the loss using the formula

$$L = (4\pi R/\lambda)^2$$

```
R = sqrt((pos1-pos2)'*(pos1-pos2));
lambda = physconst('Lightspeed')/fop;
L = (4*pi*R/lambda)^2
```

```
L = 2.4924e+11
```

Because the transmitted amplitude is unity, the magnitude-squared value of the signal at the target for the third sample equals the inverse of the loss.

```
disp(1/abs(y(3))^2)
```

```
   2.4924e+11
```

Signal Propagation from Moving Radar to Moving Target

Calculate the result of propagating a signal in free space from a radar at (1000,0,0) to a target at (300,200,50). Assume the radar moves at 10 m/s along the *x*-axis, while the target moves at 15 m/s along the *y*-axis. The sample rate is 8000 Hz while the operating frequency of the radar is 300 MHz. The signal propagation speed takes the default value of the speed of light. Transmit five samples of a unit amplitude signal and examine the amplitude of the signal at the target.

```
fs = 8000;
fop = 3e8;
freesp = phased.FreeSpace(SampleRate=fs, ...
    OperatingFrequency=fop);
pos1 = [1000;0;0];
pos2 = [300;200;50];
vel1 = [10;0;0];
vel2 = [0;15;0];
y = freesp(ones(5,1),pos1,pos2,vel1,vel2);
disp(y)
```

```
   1.0e-03 *

   0.0126 - 0.1061i
   0.0117 - 0.1083i
   0.0105 - 0.1085i
   0.0094 - 0.1086i
   0.0082 - 0.1087i
```

Because the transmitted amplitude is unity, the square of the signal at the target equals the inverse of the loss.

```
disp(1/abs(y(2))^2)
```

```
    8.4206e+07
```

Propagation of Polarized Field from Source to Target

Create a uniform linear array (ULA) consisting of four short-dipole antenna elements that support polarization. Set the orientation of each dipole to the z-direction. Set the operating frequency to 300 MHz and the element spacing of the array to 0.4 meters. While the antenna element supports polarization, you must explicitly enable polarization in the Radiator System object™.

Create the short-dipole antenna element, ULA array, and radiator System objects. Set the `CombineRadiatedSignals` property to `true` to coherently combine the radiated signals from all antennas and the `Polarization` property to `'Combined'` to process polarized waves.

```
freq = 300e6;
nsensors = 4;
c = physconst('LightSpeed');
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 900e6],...
    'AxisDirection','Z');
array = phased.ULA('Element',antenna,...
    'NumElements',nsensors,...
    'ElementSpacing',0.4);
radiator = phased.Radiator('Sensor',array,...
    'PropagationSpeed',c,...
    'OperatingFrequency',freq,...
    'CombineRadiatedSignals',true,...
    'Polarization','Combined',...
    'WeightsInputPort',true);
```

Create a signal to be radiated. In this case, the signal consists of one cycle of a 4 kHz sinusoid. Set the signal amplitude to unity. Set the sampling frequency to 8 kHz. Choose radiating angles of 0 degrees azimuth and 20 degrees elevation. For polarization, you must set a local axes - in this case chosen to coincide with the global axes. Set uniform weights on the elements of the array.

```
fsig = 4000;
fs = 8000;
A = 1;
t = [0:0.01:2]/fs;
signal = A*sin(2*pi*fsig*t');
radiatingAngles = [0;20];
laxes = ones(3,3);
y = radiator(signal,radiatingAngles,laxes,[1,1,1,1].');
disp(y)
```

```
    X: [201x1 double]
    Y: [201x1 double]
    Z: [201x1 double]
```

The radiated signal is a `struct` containing the polarized field.

Use a `FreeSpace` System object to propagate the field from the origin to the destination.

```
propagator = phased.FreeSpace('PropagationSpeed',c,...
    'OperatingFrequency',freq,...
    'TwoWayPropagation',false,...
    'SampleRate',fs);
```

Set the signal origin, signal origin velocity, signal destination, and signal destination velocity.

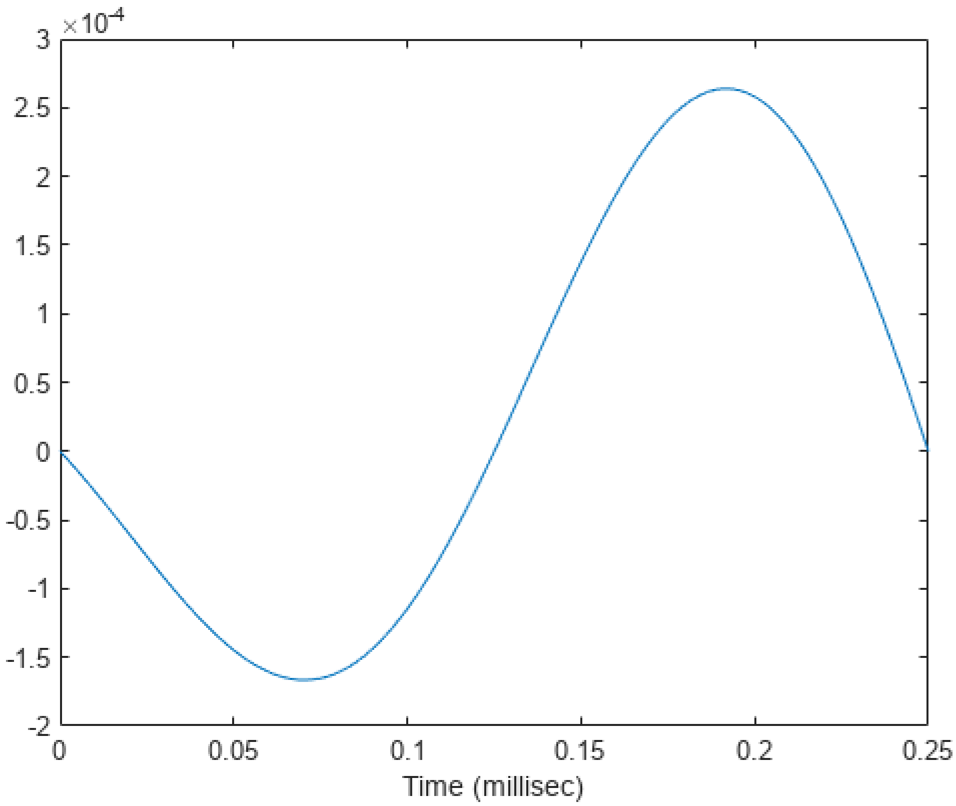
```
origin_pos = [0; 0; 0];
dest_pos = [500; 200; 50];
origin_vel = [10; 0; 0];
dest_vel = [0; 15; 0];
```

Call the FreeSpace object to propagate the signals.

```
yprop = propagator(y,origin_pos,dest_pos,...
    origin_vel,dest_vel);
```

Plot the x-component of the propagated signals.

```
figure
plot(1000*t,real(yprop.X))
xlabel('Time (millisec)')
```



Propagate Signal to Multiple Destinations

Create a FreeSpace System object™ to propagate a signal from one point to multiple points in space. Start by defining a signal origin and three destination points, all at different ranges.

Compute the propagation direction angles from the source to the destination locations. The source and destination are stationary.

```
pos1 = [0,0,0]';
vel1 = [0,0,0]';
pos2 = [[700;700;100],[1400;1400;200],2*[2100;2100;400]];
vel2 = zeros(size(pos2));
[rngs,radiatingAngles] = rangeangle(pos2,pos1);
```

Create the cosine antenna element, ULA array, and Radiator System objects.

```
fs = 8000;
freq = 300e6;
nsensors = 4;
sAnt = phased.CosineAntennaElement;
sArray = phased.ULA('Element',sAnt,'NumElements',nsensors);
sRad = phased.Radiator('Sensor',sArray,...
    'OperatingFrequency',freq,...
    'CombineRadiatedSignals',true,'WeightsInputPort',true);
```

Create a signal to be one cycle of a sinusoid of amplitude one and having a frequency of 4 kHz.

```
fsig = 4000;
t = [0:0.01:2]'/fs;
signal = sin(2*pi*fsig*t);
```

Radiate the signals in the destination directions. Apply a uniform weighting to the array.

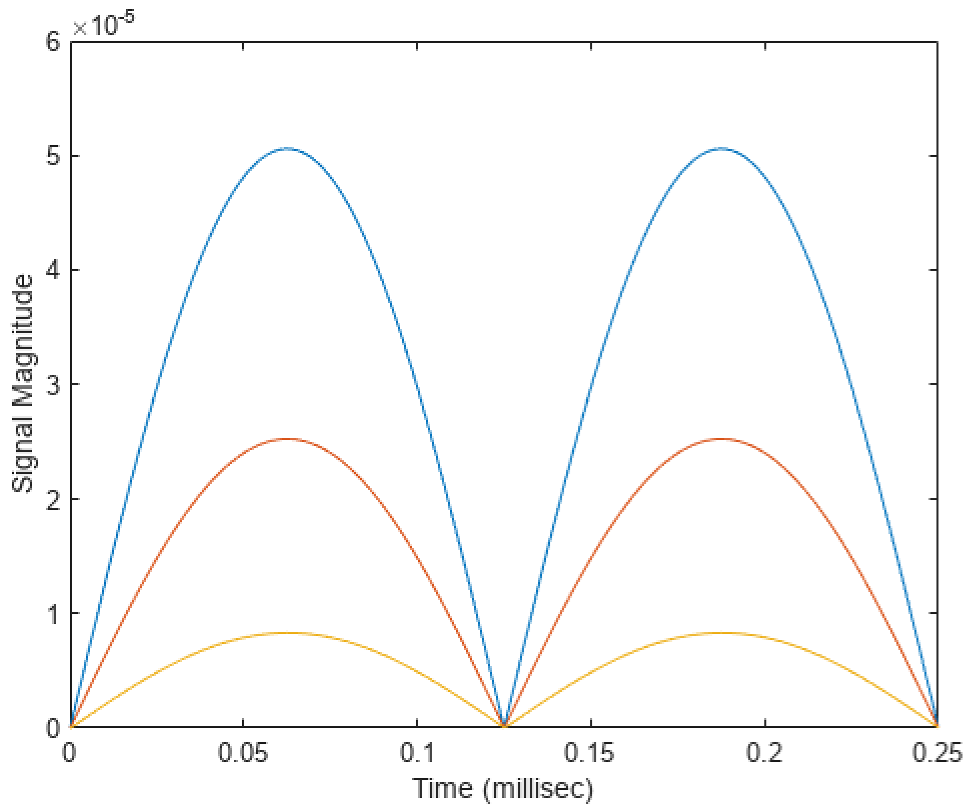
```
y = step(sRad,signal,radiatingAngles,[1,1,1,1].');
```

Propagate the signals to the destination points.

```
sFSp = phased.FreeSpace('OperatingFrequency',freq,'SampleRate',fs);
yprop = step(sFSp,y,pos1,pos2,vel1,vel2);
```

Plot the propagated signal magnitudes for each range.

```
figure
plot(1000*t,abs(yprop(:,1)),1000*t,abs(yprop(:,2)),1000*t,abs(yprop(:,3)))
ylabel('Signal Magnitude')
xlabel('Time (millisec)')
```



Algorithms

When the origin and destination are stationary relative to each other, you can write the output signal of a free-space channel as $Y(t) = x(t-\tau)/L_{fsp}$. The quantity τ is the signal delay and L_{fsp} is the free-space path loss. The delay τ is given by R/c, where R is the propagation distance and c is the propagation speed. The free-space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far field of the transmitting element or array. In the near field, the free-space path loss formula is not valid and can result in a loss smaller than one, equivalent to a signal gain. Therefore, the loss is set to unity for range values, $R \leq \lambda/4\pi$.

When the origin and destination have relative motion, the processing also introduces a Doppler frequency shift. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The quantity v is the relative speed of the destination with respect to the origin.

For further details, see [2].

References

[1] Proakis, J. *Digital Communications*. New York: McGraw-Hill, 2001.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

See Also

`twoRayChannel` | `phased.WidebandFreeSpace`

phased.FrostBeamformer

Package: phased

Frost beamformer

Description

The phased.FrostBeamformer object implements a Frost beamformer. A Frost beamformer consists of a time-domain MVDR beamformer combined with a bank of FIR filters. The beamformer steers the beam towards a given direction while the FIR filters preserve the input signal power.

To compute the beamformed signal:

1. Create the phased.FrostBeamformer object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
beamformer = phased.FrostBeamformer
beamformer = phased.FrostBeamformer(Name,Value)
```

Description

beamformer = phased.FrostBeamformer creates a Frost beamformer System object, beamformer, with default property values.

beamformer = phased.FrostBeamformer(Name,Value) creates a Frost beamformer object, beamformer, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN). Enclose each property name in single quotes.

Example: beamformer = phased.FrostBeamformer('SensorArray',phased.ULA('NumElements',20),'SampleRate',300e3) sets the sensor array to a uniform linear array (ULA) with default ULA property values except for the number of elements. The beamformer has a sample rate of 300 kHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

SensorArray — Sensor array
phased.ULA array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object. The array cannot contain subarrays.

Example: `phased.URA`

PropagationSpeed — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`.

Example: `3e8`

Data Types: `single` | `double`

SampleRate — Sample rate of signal
`1e6` (default) | positive scalar

Sample rate of signal, specified as a positive scalar. Units are in Hz. The System object uses this quantity to calculate the propagation delay in units of samples.

Example: `1e6`

Data Types: `single` | `double`

FilterLength — FIR filter length
`2` (default) | positive integer

Length of FIR filter for each sensor element, specified as a positive integer.

Example: `7`

Data Types: `single` | `double`

DiagonalLoadingFactor — Diagonal loading factor
`0` (default) | nonnegative scalar

Diagonal loading factor, specified as a nonnegative scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample size is small. A small sample size can lead to an inaccurate estimate of the covariance matrix. Diagonal loading also provides robustness due to steering vector errors. The diagonal loading technique adds a positive scalar multiple of the identity matrix to the sample covariance matrix.

Tunable: Yes

Data Types: `single` | `double`

TrainingInputPort — Enable training data input
`false` (default) | `true`

Enable training data input, specified as `false` or `true`. When you set this property to `true`, use the training data input argument, `XT`, when running the object. Set this property to `false` to use the input data, `X`, as the training data.

Data Types: `logical`

DirectionSource — Source of beamforming direction
'Property' (default) | 'Input port'

Source of beamforming direction, specified as 'Property' or 'Input port'. Specify whether the beamforming direction comes from the Direction property of this object or from the input argument, ANG. Values of this property are:

'Property'	Specify the beamforming direction using the Direction property.
'Input port'	Specify the beamforming direction using the input argument, ANG.

Data Types: char

Direction — Beamforming directions
[0;0] (default) | real-valued 2-by-1 vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 vector or a real-valued 2-by-*L* matrix. For a matrix, each column specifies a different beamforming direction. Each column has the form [AzimuthAngle;ElevationAngle]. Azimuth angles must lie between –180° and 180° and elevation angles must lie between –90° and 90°. All angles are defined with respect to the local coordinate system of the array. Units are in degrees.

Example: [40;30]

Dependencies

To enable this property, set the DirectionSource property to 'Property'.

Data Types: single | double

WeightsOutputPort — Enable beamforming weights output
false (default) | true

Enable the output of beamforming weights, specified as false or true. To obtain the beamforming weights, set this property to true and use the corresponding output argument, W. If you do not want to obtain the weights, set this property to false.

Data Types: logical

Usage

Syntax

```
Y = beamformer(X)
Y = beamformer(X,XT)
Y = beamformer(X,ANG)
Y = beamformer(X,XT,ANG)
[Y,W] = beamformer(___)
```

Description

Y = beamformer(X) performs Frost beamforming on the input, X, and returns the beamformed output, Y. This syntax uses the input data, X, as training samples to calculate the beamforming weights.

`Y = beamformer(X,XT)` uses `XT` as training data to calculate the beamforming weights. To use this syntax, set the TrainingInputPort property to `true`.

`Y = beamformer(X,ANG)` uses `ANG` as the beamforming direction. To use this syntax, set the DirectionSource property to `'Input port'`.

`Y = beamformer(X,XT,ANG)` combines all input arguments. To use this syntax, set the TrainingInputPort property to `true` and set the DirectionSource property to `'Input port'`.

`[Y,W] = beamformer( ___ )` returns the beamforming weights, `W`. To use this syntax, set the WeightsOutputPort property to `true`.

Input Arguments

X — Input signal
complex-valued *M*-by-*N* matrix

Input signal, specified as a complex-valued *M*-by-*N* matrix. *M* is the signal length and *N* is the number of array elements specified in the SensorArray property. *M* must be larger than the length of the filter specified by the FilterLength property.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`
Complex Number Support: Yes

XT — Training data
complex-valued *M*-by-*N* matrix

Training data, specified as a complex-valued *M*-by-*N* matrix. *M* and *N* are equal to the values for `X`.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1 0.5 2.6; 2 -0.2 0; 3 -2 -1]`

Dependencies

To enable this argument, set the TrainingInputPort property to `true`.

Data Types: `single` | `double`
Complex Number Support: Yes

ANG — Beamforming directions
`[0;0]` (default) | real-valued 2-by-1 column vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 column vector The vector has the form `[AzimuthAngle;ElevationAngle]`. Units are in degrees. The azimuth angle must lie between –180° and 180°, and the elevation angle must lie between –90° and 90°.

Example: `[40;10]`

Dependencies

To enable this argument, set the DirectionSource property to `'Input port'`.

Data Types: `double`

Output Arguments

Y — Beamformed output
complex-valued 1-by-*M* vector

Beamformed output, returned as a complex-valued 1-by-*M*vector, where *M* is the number of rows of the input `X`.

Data Types: `single` | `double`
Complex Number Support: Yes

W — Beamforming weights
complex-valued *L*-by-1 vector.

Beamforming weights, returned as a complex-valued *L*-by-1 vector where *L* is the number of degrees of freedom of the beamformer. The number of degrees of freedom is given by the product of the number of elements specified by the SensorArray property and the FIR filter length specified by FilterLength property.

Dependencies

To enable this output, set the WeightsOutputPort property to `true`.

Data Types: `single` | `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Apply Frost Beamforming to ULA

Apply Frost beamforming to an 11-element acoustic ULA array. The incident angle of the incoming signal is -50 degrees in azimuth and 30 degrees in elevation. The speed of sound in air is assumed to be 340 m/sec. The signal has added Gaussian white noise.

Simulate the signal.

```
array = phased.ULA('NumElements',11,'ElementSpacing',0.04);
array.Element.FrequencyRange = [20 20000];
```

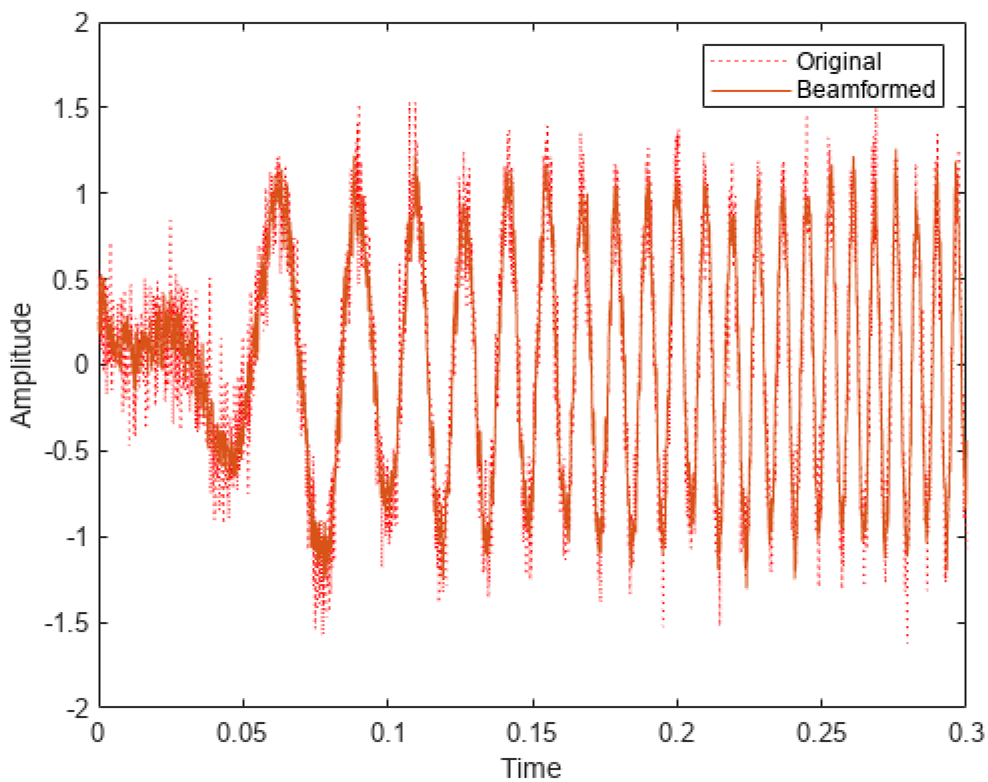
```
fs = 8e3;
t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340;
collector = phased.WidebandCollector('Sensor',array,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',false,'NumSubbands',8192);
incidentAngle = [-50;30];
x = collector(x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x + noise;
```

Beamform the signal.

```
beamformer = phased.FrostBeamformer('SensorArray',array,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'Direction',incidentAngle,'FilterLength',5);
y = beamformer(rx);
```

Plot the beamformed output.

```
plot(t,rx(:,6),'r:',t,y)
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Find Frost Beamforming Weights

Find the beamformer weights of a Frost beamforming applied to signals received at a 7-element acoustic ULA array. The incident angle of the incoming signal is -20° in azimuth and 30° in elevation. The signal has added Gaussian white noise. The speed of sound in air is assumed to be 340 m/s. Use a filter length of 15.

First, create the signal.

```
numelements = 7;
element = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[50,10000]);
array = phased.ULA('Element',element,'NumElements',numelements,'ElementSpacing',0.04);
fs = 8e3;
t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340.0;
collector = phased.WidebandCollector('Sensor',array,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',false,'NumSubbands',8192);
incidentAngle = [-20;30];
x = collector(x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x + noise;
```

Create a beamformer with a filter length of 15. Then, beamform the arriving signal and obtain the beamformer weights.

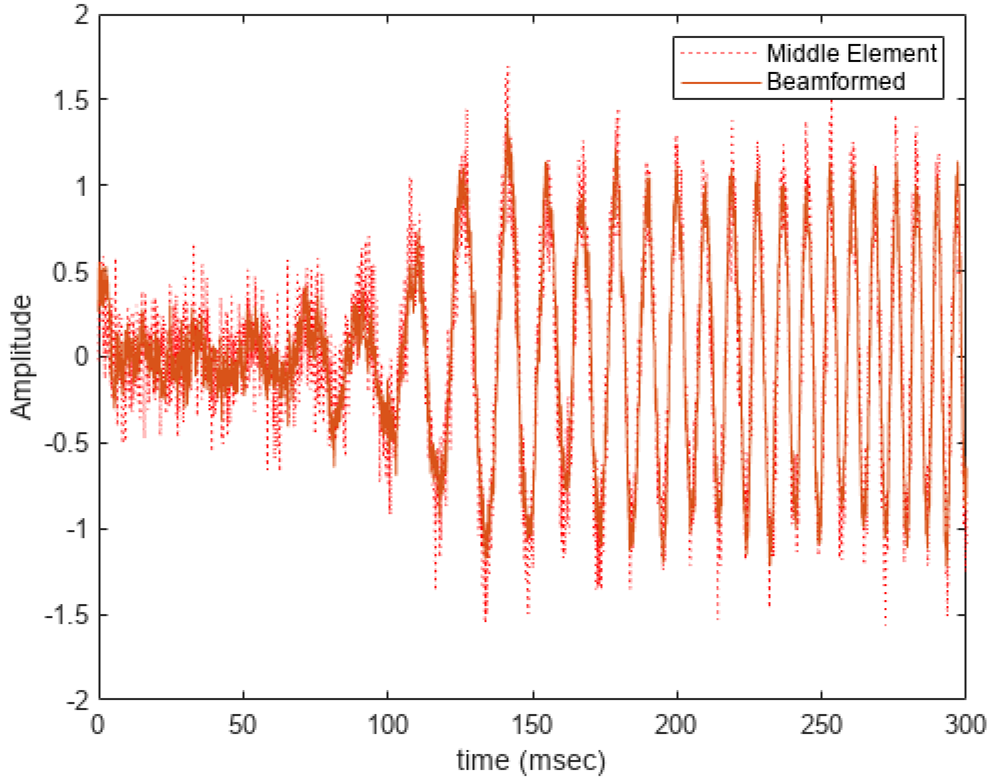
```
filterlength = 15;
beamformer = phased.FrostBeamformer('SensorArray',array, ...
    'PropagationSpeed',c,'SampleRate',fs,'WeightsOutputPort',true, ...
    'Direction',incidentAngle,'FilterLength',filterlength);
[y,wt] = beamformer(rx);
size(wt)
```

```
ans = 1×2

   105     1
```

There are 7*15 = 105 weights computed as expected.

Compare the beamformed output with the signal arriving at the middle element of the array.

```
plot(1000*t,rx(:,4),'r:',1000*t,y)
xlabel('time (msec)')
ylabel('Amplitude')
legend('Middle Element','Beamformed')
```



Algorithms

`phased.FrostBeamformer` uses a beamforming algorithm proposed by Frost. It can be considered the time-domain counterpart of the minimum variance distortionless response (MVDR) beamformer. The algorithm does the following:

1. Steers the array to the beamforming direction.
2. Applies an FIR filter to the output of each sensor to achieve the distortionless response constraint. The filter is specific to each sensor.
3. This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

For more information about Frost beamforming, see [1].

Version History

Introduced in R2011a

References

[1] Frost, O. “An Algorithm For Linearly Constrained Adaptive Array Processing”, *Proceedings of the IEEE*. Vol. 60, Number 8, August, 1972, pp. 926–935.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Requires dynamic memory allocation. See “Limitations for System Objects that Require Dynamic Memory Allocation”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also
`phased.PhaseShiftBeamformer` | `phased.SubbandPhaseShiftBeamformer` | `phased.TimeDelayBeamformer` | `phased.TimeDelayLCMVBeamformer`

step

System object: `phased.FrostBeamformer`
Package: `phased`

Perform Frost beamforming

Syntax

```
Y = step(H,X)
Y = step(H,X,XT)
Y = step(H,X,ANG)
Y = step(H,X,XT,ANG)
[Y,W] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` performs Frost beamforming on the input, `X`, and returns the beamformed output in `Y`.

`Y = step(H,X,XT)` uses `XT` as the training samples to calculate the beamforming weights. This syntax is available when you set the `TrainingInputPort` property to `true`.

`Y = step(H,X,ANG)` uses `ANG` as the beamforming direction. This syntax is available when you set the `DirectionSource` property to `'Input port'`.

`Y = step(H,X,XT,ANG)` combines all input arguments. This syntax is available when you set the `TrainingInputPort` property to `true` and set the `DirectionSource` property to `'Input port'`.

`[Y,W] = step( ___ )` returns the beamforming weights, `W`. This syntax is available when you set the `WeightsOutputPort` property to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Beamformer object.

X

Input signal, specified as an *M*-by-*N* matrix. *M* must be larger than the FIR filter length specified in the `FilterLength` property. *N* is the number of elements in the sensor array.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

XT

Training samples, specified as an *M*-by-*N* matrix. *M* and *N* have the same dimensions as `X`.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

ANG

Beamforming directions, specified as a length-2 column vector. The vector has the form `[AzimuthAngle; ElevationAngle]`, in degrees. The azimuth angle must be between –180 and 180 degrees, and the elevation angle must be between –90 and 90 degrees.

Output Arguments

Y

Beamformed output. `Y` is a column vector of length *M*, where *M* is the number of rows in `X`.

W

Beamforming weights. `W` is a column vector of length *L*, where *L* is the degrees of freedom of the beamformer. For a Frost beamformer, `H`, *L* is given by the product of the number of elements in the array and the FIR filter length.

Examples

Apply Frost Beamforming to ULA

Apply Frost beamforming to an 11-element acoustic ULA array. The incident angle of the incoming signal is -50 degrees in azimuth and 30 degrees in elevation. The speed of sound in air is assumed to be 340 m/sec. The signal has added Gaussian white noise.

Simulate the signal.

```
array = phased.ULA('NumElements',11,'ElementSpacing',0.04);
array.Element.FrequencyRange = [20 20000];
fs = 8e3;
t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340;
collector = phased.WidebandCollector('Sensor',array,...
    'PropagationSpeed',c,'SampleRate',fs,...
```

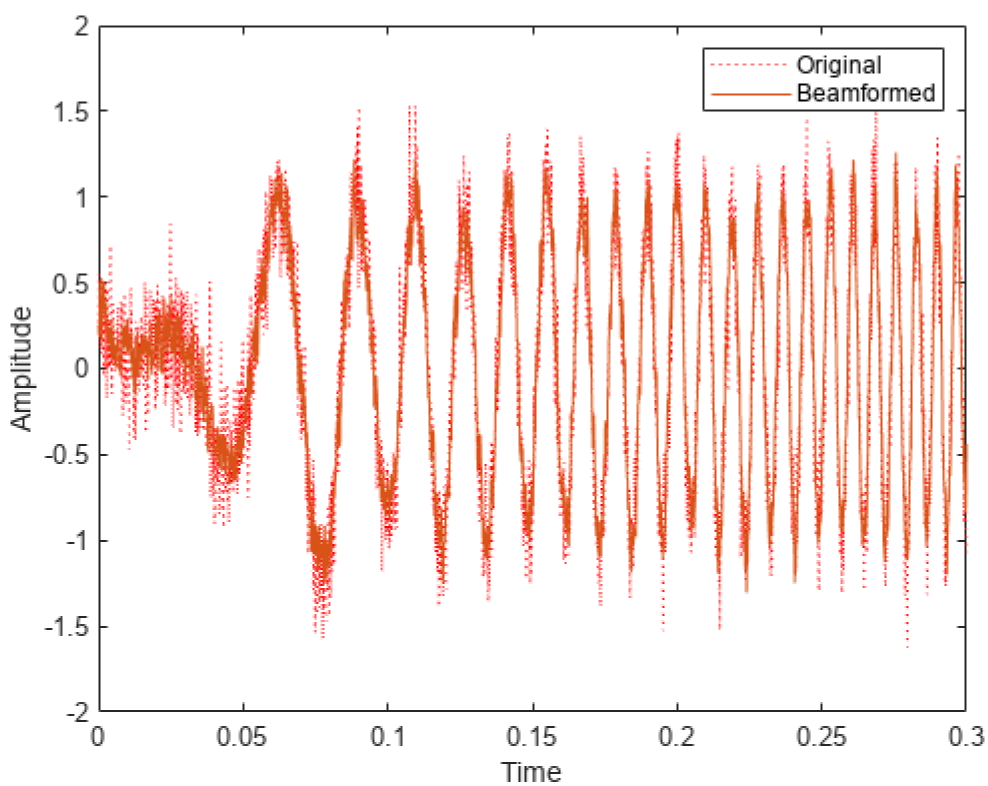
```
    'ModulatedInput',false,'NumSubbands',8192);
incidentAngle = [-50;30];
x = collector(x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x + noise;
```

Beamform the signal.

```
beamformer = phased.FrostBeamformer('SensorArray',array,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'Direction',incidentAngle,'FilterLength',5);
y = beamformer(rx);
```

Plot the beamformed output.

```
plot(t,rx(:,6),'r:',t,y)
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Algorithms

`phased.FrostBeamformer` uses a beamforming algorithm proposed by Frost. It can be considered the time-domain counterpart of the minimum variance distortionless response (MVDR) beamformer. The algorithm does the following:

1. Steers the array to the beamforming direction.
2. Applies an FIR filter to the output of each sensor to achieve the distortionless response constraint. The filter is specific to each sensor.

For further details, see [1].

References

[1] Frost, O. “An Algorithm For Linearly Constrained Adaptive Array Processing”, *Proceedings of the IEEE*. Vol. 60, Number 8, August, 1972, pp. 926–935.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

phased.FocusedSteeringVector

Package: `phased`

Focused sensor array steering vector

Description

The `phased.FocusedSteeringVector` System object creates steering vectors for focusing a sensor array at a specified point at multiple ranges, directions, and frequencies.

To compute a focused steering vector for an array

1. Create the `phased.FocusedSteeringVector` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
fsteervec = phased.FocusedSteeringVector
fsteervec = phased.FocusedSteeringVector(Name,Value)
```

Description

`fsteervec = phased.FocusedSteeringVector` creates a focused steering vector System object `fsteervec` with default property values.

`fsteervec = phased.FocusedSteeringVector(Name,Value)` creates a focused steering vector System object with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `focusedsv = phased.FocusedSteeringVector('SensorArray',phased.URA([10,20],'ElementSpacing',[0.25,0.25]),'PropagationSpeed',physconst('LightSpeed'))` creates a focused steering vector object for a 10-by-20 uniform rectangular array (URA) with element spacing set to 0.25 meters The propagation speed is set to the speed of light.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

SensorArray — Sensor array
phased.ULA array with default property values (default) | Phased Array System Toolbox array

Sensor array, specified as an array System object belonging to Phased Array System Toolbox. The sensor array can contain subarrays.

Example: phased.URA

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: 3e8

Data Types: double

IncludeElementResponse — Include individual element responses in the steering vector
false (default) | true

Option to include individual element responses in the steering vector, specified as `false` or `true`. If this property is set to `true`, the steering vector includes individual array element responses. If this property is set to `false`, the steering vector is computed assuming that the elements are isotropic, regardless of how the elements are specified. Set this property to true when using polarized signals.

When the array specified in the SensorArray property contains subarrays, the steering vector applies to subarrays. If SensorArray does not contain subarrays, the steering vector applies to the array elements.

Data Types: logical

NumPhaseShifterBits — Number of phase shifter quantization bits
0 (default) | nonnegative integer

Number of phase shifter quantization bits, specified as a nonnegative integer. This number of bits is used to quantize the phase shift component of the beamformer or steering vector weights. A value of zero indicates that no quantization is performed.

Data Types: double

EnablePolarization — Enable polarized fields
false (default) | true

Option to enable polarized fields, specified as `false` or `true`. Set this property to `true` to enable polarization. Set this property to `false` to ignore polarization. Enabling polarization requires that the sensor array specified in the SensorArray property can simulate polarization.

If you set this property to `false` for an array that actually supports polarization, then all polarization information is discarded. A combined pattern from the *H* and *V* polarization components is used at each element to compute the steering vector.

Data Types: logical

Usage

Syntax

```
FSV = fsteervec(FREQ,ANG,FOC)
[FSV,RANGE] = fsteervec(FREQ,ANG,FOC)
```

Description

`FSV = fsteervec(FREQ,ANG,FOC)` returns the focused steering vector, `FSV`, pointing in the directions specified by `ANG`, for range `FOC`, and for the operating frequencies specified in `FREQ`. The meaning of `FSV` depends on the `IncludeElementResponse` property, as follows:

- When `IncludeElementResponse` is `true`, the components of `FSV` include individual element responses.
- When `IncludeElementResponse` is `false`, the computation assumes that the elements are isotropic and `FSV` does not include the individual element responses. If the array contains subarrays, `FSV` is the array factor among the subarrays. The phase center of each subarray is at its geometric center. If `SensorArray` does not contain subarrays, `FSV` is the array factor among the elements.

`[FSV,RANGE] = fsteervec(FREQ,ANG,FOC)` also returns the range `RANGE` from the focus point to each antenna element.

Input Arguments

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

ANG — Angle focal point direction
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angle focal point direction, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: `[45 60; 0 10]`

Data Types: `double`

FOC — Focal range
positive scalar | *M*-element row vector of positive values

Focal range, specified as a scalar or an *M*-element row vector of positive values. If `ANG` has more than one column, `FOC` must be a scalar or have the same number of columns as `ANG`. Units are in meters.

Data Types: `double`

Output Arguments

FSV — Focused steering vector
complex-valued *N*-by-*M*-by-*L* array | structures

Focused steering vector, returned as a complex-valued *N*-by-*M*-by-*L* array or a structure containing arrays.

The form of the steering vector depends upon whether the `EnablePolarization` property is set to `true` or `false`.

- If `EnablePolarization` is set to `false`, the steering vector, `FSV`, is an *N*-by-*M*-by-*L* array. The length of the first dimension, *N*, is the number of elements of the phased array. If `SensorArray` contains subarrays, *N* is the number of subarrays. The length of the second dimension, *M*, corresponding to the number of steering directions specified in the `ANG` argument. The length of the third dimension, *L*, is the number of frequencies specified in the `FREQ` argument.
- If `EnablePolarization` is set to `true`, `SV` is a MATLAB `struct` containing two fields, `SV.H` and `SV.V`. These two fields represent the horizontal (*H*) and vertical (*V*) polarization components of the steering vector. Each field is an *N*-by-*M*-by-*L* array. The length of the first dimension, *N*, is the number of elements of the phased array. If `SensorArray` contains subarrays, *N* is the number of subarrays. The length of the second dimension, *M*, corresponds to the number of steering directions specified in the `ANG` argument. The length of the third dimension, *L*, is the number of frequencies specified in the `FREQ` argument.

  Simulating polarization also requires that the sensor array specified in the `SensorArray` property can simulate polarization, and that the `IncludeElementResponse` property is set to `true`.

Data Types: `double`

RANGE — Range from array elements to array focus
N-by-*M* real-valued array of positive values

Range from array elements to array focus, returned as an *N*-by-*M* real-valued array of positive values. Units are in meters.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Focus ULA at Specified Point

Compute the focused steering vector of an 11-element uniform linear array. Focus the array in the direction of 45° azimuth and 30° elevation and at a range of 1 km. Assume the array operating frequency is 300 MHz and the elements are spaced 0.4 wavelengths apart.

```
fc = 300.0e6;
c = physconst('lightspeed');
az = 45.0;
el = 30.0;
foc = 1000.0;
lambda = c/fc;
elementspacing = 0.4*lambda;
nelem = 11;
array = phased.ULA(nelem,elementspacing);
fsteervec = phased.FocusedSteeringVector('SensorArray',array);
fsvec = fsteervec(fc,[az;el],foc);
```

Compute Range of Element to Focus of URA

Compute the range from the focal point of a focused 3-by-4 URA to the array elements. The array focuses at a point 1000 m away in the direction of 45° elevation and 30° azimuth. The array operating frequency is 300 Mhz and the array elements are spaced 1/2 wavelength apart.

```
fc = 300.0e6;
c = physconst('lightspeed');
az = 45.0;
el = 30.0;
foc = 1000.0;
lambda = c/fc;
elementspacing = 0.5*lambda;
```

Find the steering vector and element ranges.

```
array = phased.URA([3,4],elementspacing);
fsteervec = phased.FocusedSteeringVector('SensorArray',array);
```

```
[fsvec,elemrng] = fsteervec(fc,[az;el],foc);
disp(elemrng)
```

```
   1.0e+03 *

    1.0002
    1.0005
    1.0007
    0.9999
    1.0002
    1.0004
    0.9996
    0.9998
    1.0001
    0.9993
    0.9995
    0.9998
```

Version History

Introduced in R2021b

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.ArrayGain` | `phased.ArrayResponse` | `phased.ElementDelay` | `phased.SteeringVector` | `phased.SphericalWavefrontArrayResponse`

Topics

"Examine the Response of a Focused Phased Array"

phased.GCCEstimator

Package: `phased`

Wideband direction of arrival estimation

Description

The `phased.GCCEstimator` System object creates a direction of arrival estimator for wideband signals. This System object estimates the direction of arrival or time of arrival among sensor array elements using the generalized cross-correlation with phase transform algorithm (GCC-PHAT). The algorithm assumes that all signals propagate from a single source lying in the array far field so the direction of arrival is the same for all sensors. The System object first estimates the correlations between all specified sensor pairs using GCC-PHAT and then finds the largest peak in each correlation. The peak identifies the delay between the signals arriving at each sensor pair. Finally, a least-squares estimate is used to derive the direction of arrival from all estimated delays.

To compute the direction of arrival for pairs of element in the array:

1. Define and set up a GCC-PHAT estimator System object, `phased.GCCEstimator`, using the "Construction" on page 1-469 procedure.
2. Call `step` to compute the direction of arrival of a signal using the properties of the `phased.GCCEstimator` System object.

   The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`sGCC = phased.GCCEstimator` creates a GCC direction of arrival estimator System object, `sGCC`. This object estimates the direction of arrival or time of arrival between sensor array elements using the GCC-PHAT algorithm.

`sGCC = phased.GCCEstimator(Name,Value)` returns a GCC direction of arrival estimator object, `sGCC`, with the specified property `Name` set to the specified `Value`. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Properties

`SensorArray` — Sensor array

`phased.ULA` System object (default) | Phased Array System Toolbox sensor array

Sensor array, specified as a Phased Array System Toolbox System object. The array can also consist of subarrays. If you do not specify this property, the default sensor array is a `phased.ULA` System object with default array property values.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`.

Example: `3e8`

Data Types: `single` | `double`

`SampleRate` — Signal sample rate
`1e6` (default) | positive real-valued scalar

Signal sample rate, specified as a positive real-valued scalar. Units are in hertz.

Example: `1e6`

Data Types: `single` | `double`

`SensorPairSource` — Source of sensor pairs
`'Auto'` (default) | `'Property'`

Source of sensor pairs, specified as either `'Auto'` or `'Property'`.

- `'Auto'` — choose this property value to compute correlations between the first sensor and all other sensors. The first sensor of the array is the reference channel.
- `'Property'` — choose this property value when you want to explicitly specify the sensor pairs to be used for computing correlations. Set the sensor pair indices using the `SensorPair` property. You can view the array indices using the `viewArray` method of any array System object.

Example: `'Auto'`

Data Types: `char`

`SensorPair` — Sensor pairs
`[2;1]` (default) | 2-by-*N* positive integer valued matrix

Sensor pairs used to compute correlations, specified as a 2-by-*N* positive integer-valued matrix. Each column of the matrix specifies a pair of sensors between which the correlation is computed. The second row specifies the reference sensors. Each entry in the matrix must be less than the number of array sensors or subarrays. To use the `SensorPair` property, you must also set the `SensorPairSource` value to `'Property'`.

Example: `[1,3,5;2,4,6]`

Data Types: `double`

`DelayOutputPort` — Option to enable delay output
`false` (default) | `true`

Option to enable output of time delay values, specified as a Boolean. Set this property to `true` to output the delay values as an output argument of the `step` method. The delays correspond to the arrival angles of a signal between sensor pairs. Set this property to `false` to disable the output of delays.

Example: `false`

Data Types: `logical`

`CorrelationOutputPort` — Option to enable correlation output
`false` (default) | `true`

Option to enable output of correlation values, specified as a Boolean. Set this property to `true` to output the correlations and lags between sensor pairs as output arguments of the `step` method. Set this property to `false` to disable output of correlations.

Example: false

Data Types: `logical`

Methods

reset	Reset states of phased.GCCEstimator System object
step	Estimate direction of arrival using generalized cross-correlation

Common to All System Objects	
`release`	Allow System object property value changes

Examples

GCC Estimate of Direction of Arrival at Microphone Array

Estimate the direction of arrival of a signal using the GCC-PHAT algorithm. The receiving array is a 5-by-5-element URA microphone array with elements spaced 0.25 meters apart. The arriving signal is a sequence of wideband chirps. The signal arrives from 17° azimuth and 0° elevation. Assume the speed of sound in air is 340 m/s.

Load the chirp signal.

```
load chirp;
c = 340.0;
```

Create the 5-by-5 microphone URA.

```
d = 0.25;
N = 5;
mic = phased.OmnidirectionalMicrophoneElement;
array = phased.URA([N,N],[d,d],'Element',mic);
```

Simulate the incoming signal using the `WidebandCollector` System object™.

```
arrivalAng = [17;0];
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c,...
    'SampleRate',Fs,'ModulatedInput',false);
signal = collector(y,arrivalAng);
```

Estimate the direction of arrival.

```
estimator = phased.GCCEstimator('SensorArray',array,...
    'PropagationSpeed',c,'SampleRate',Fs);
ang = estimator(signal)
```

```
ang = 2×1

   16.4538
   -0.7145
```

Algorithms

GCC-PHAT Cross-Correlation Algorithm

You can use generalized cross-correlation to estimate the time difference of arrival of a signal at two different sensors.

A model of a signal emitted by a source and received at two sensors is given by:

$$r_1(t) = s(t) + n_1(t)$$
$$r_2(t) = s(t - D) + n_2(t)$$

where D is the time difference of arrival (*TDOA*), or time lag, of the signal at one sensor with respect to the arrival time at a second sensor. You can estimate the time delay by finding the time lag that maximizes the cross-correlation between the two signals.

From the TDOA, you can estimate the broadside arrival angle of the plane wave with respect to the line connecting the two sensors. For two sensors separated by distance L, the broadside arrival angle, “Broadside Angles”, is related to the time lag by

$$\sin\beta = \frac{c\tau}{L}$$

where c is the propagation speed in the medium.

A common method of estimating time delay is to compute the cross-correlation between signals received at two sensors. To identify the time delay, locate the peak in the cross-correlation. When the signal-to-noise ratio (SNR) is large, the correlation peak, τ, corresponds to the actual time delay D.

$$R(\tau) = E\{r_1(t)r_2(t + \tau)\}$$
$$\widehat{D} = \underset{\tau}{\operatorname{argmax}} R(\tau)$$

When the correlation function is more sharply peaked, performance improves. You can sharpen a cross correlation peak using a weighting function that whitens the input signals. This technique is called generalized cross-correlation (GCC). One particular weighting function normalizes the signal spectral density by the spectrum magnitude, leading to the generalized cross-correlation phase transform method (*GCC-PHAT*).

$$S(f) = \int_{-\infty}^{\infty} R(\tau)e^{-i2\pi f\tau}d\tau$$
$$\tilde{R}(\tau) = \int_{-\infty}^{\infty} \frac{S(f)}{|S(f)|}e^{+i2\pi f\tau}df$$
$$\tilde{D} = \underset{\tau}{\operatorname{argmax}}\ \tilde{R}(\tau)$$

If you use just one sensor pair, you can only estimate the broadside angle of arrival. However, if you use multiple pairs of non-collinear sensors, for example, in a URA, you can estimate the arrival azimuth and elevation angles of the plane wave using least-square estimation. For N sensors, you can write the delay time τ_{kj} of a signal arriving at the k^{th} sensor with respect to the j^{th} sensor by

$$c\tau_{kj} = -\left(\vec{x}_k - \vec{x}_j\right) \cdot \vec{u}$$
$$\vec{u} = \cos\alpha\sin\theta\hat{i} + \sin\alpha\sin\theta\hat{j} + \cos\theta\hat{k}$$

where u is the unit propagation vector of the plane wave. The angles α and θ are the azimuth and elevation angles of the propagation vector. All angles and vectors are defined with respect to the local axes. You can solve the first equation using least-squares to yield the three components of the unit propagation vector. Then, you can solve the second equation for the azimuth and elevation angles.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2015b

References

[1] Knapp, C. H. and G.C. Carter, “The Generalized Correlation Method for Estimation of Time Delay.” *IEEE Transactions on Acoustics, Speech and Signal Processing.* Vol. ASSP-24, No. 4, Aug 1976.

[2] G. C. Carter, “Coherence and Time Delay Estimation.” *Proceedings of the IEEE.* Vol. 75, No. 2, Feb 1987.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.BeamscanEstimator` | `phased.RootMUSICEstimator` | `gccphat`

reset

System object: `phased.GCCEstimator`
Package: `phased`

Reset states of phased.GCCEstimator System object

Syntax

`reset(S)`

Description

`reset(S)` resets the internal state of the `phased.GCCEstimator` object, `S`. This method resets the random number generator state if the `SeedSource` property is applicable and has the value `'Property'`.

Input Arguments

S — GCC-PHAT estimator
`phased.GCCEstimator` System object

GCC-PHAT estimator, specified as a `phased.GCCEstimator` System object.

Example: `phased.GCCEstimator()`

Version History

Introduced in R2015b

step

System object: `phased.GCCEstimator`
Package: `phased`

Estimate direction of arrival using generalized cross-correlation

Syntax

```
ang = step(sGCC,X)
[ang,tau] = step(sGCC,X)
[ang,R,lag] = step(sGCC,X)
[ang,tau,R,lag] = step(sGCC,X)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`ang = step(sGCC,X)` returns the direction of arrival, `ang`, of an input signal `X`. The argument `X` is a matrix specifying the received signals at the elements of the array specified in the `SensorArray` property. Signals propagate from a single source. Each column in `X` corresponds to the elements in the array (if an array is used) or the number of subarrays (if a subarray is used). Each row of `X` represents a single time snapshot.

`[ang,tau] = step(sGCC,X)` returns the time delays, `tau`, estimated from the correlations between pairs of sensors. To use this syntax, set the `DelayOutputPort` property to `true`. The argument `tau` is a *P*-element row vector, where *P* is the number of sensor pairs, and where *P* = *N(N-1)*.

`[ang,R,lag] = step(sGCC,X)` returns the estimated correlations, `R`, between pairs of sensors, when you set the `CorrelationOutputPort` property to true. `R` is a matrix with *P* columns where *P* is the number of sensor pairs. Each column in `R` contains the correlation for the corresponding pair of sensors. `lag` is a column vector containing the time lags corresponding to the rows of the correlation matrix. The time lags are the same for all sensor pairs.

You can combine optional input arguments when their enabling properties are set. Optional inputs must be listed in the same order as their enabling properties. For example,`[ang,tau,R,lag] = step(sGCC,X)` is valid when you set both `DelayOutputPort` and `CorrelationOutputPort` to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

sGCC — GCC-PHAT estimator
phased.GCCEstimator System object

GCC-PHAT estimator, specified as a `phased.GCCEstimator` System object.

Example: `phased.GCCEstimator`

X — Received signal
M-by-*N* complex-valued matrix

Received signal, specified as an *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values (snapshots) of the signal and *N* is the number of sensor elements in the array. For subarrays, *N* is the number of subarrays.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[[0;1;2;3;4;3;2;1;0],[1;2;3;4;3;2;1;0;0]]`

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

ang — Direction of arrival
2-by-1 real-valued column vector | scalar

Direction of arrival of a signal, returned as a 2-by-1 real-valued column vector in the form `[azimuth;elevation]`. If the array is a uniform linear array, `ang` is a scalar representing the broadside angle. Angle units are in degrees, defined with respect to the local coordinate system of the array.

tau — Time delays of arrival
1-by-*P* real-valued row vector

Time delays of arrival, returned as 1-by-*P* real-valued row vector. *P* is the number of sensor pairs selected from the array.

- When `SensorPairSource` is set to `'Auto'`, $P = N - 1$. *N* is the number of elements in the array.
- When `SensorPairSource` is set to `'Property'`, *P* is the number of sensor pairs specified by the `SensorPair` property.

Time units are seconds. This output is enabled when you set the `DelayOutputPort` property to `true`.

R — Estimated cross-correlation
(2*M*+1)-by-*P* complex-valued matrix

Estimated cross-correlation between pairs of sensors, returned as a (2*M*+1)-by-*P* complex-valued matrix, where *P* is the number of sensor pairs selected from the array.

- When `SensorPairSource` is set to `'Auto'`, $P = N - 1$. *N* is the number of elements in the array. The columns in `R` contain the correlations between the first sensor and all other sensors.

- When `SensorPairSource` is set to `'Property'`, *P* is the number of sensor pairs specified by the `SensorPair` property. Each column in `R` contains the correlation for the corresponding pair of sensors.

M is the number of time samples in the input signal. This output is enabled when you set the `CorrelationOutputPort` property to `true`.

lag — Time lags
M-by-1 real-valued column vector

Time lags, returned as an (2*M*+1)-by-1 real-valued column vector. *M* is the number of time samples in the input signal. Each time lag applies to the corresponding row in the cross-correlation matrix.

Examples

Plot Correlations from GCC Estimator

Estimate the direction of arrival of a signal using GCC-PHAT. The receiving array is a 5-by-5-element URA microphone array with elements spaced 25 centimeters apart. Choose 10 element pairs to compute the arrival angle. Assume the speed of sound in air is 340 meters/second. The arriving signal is a sequence of wideband sounds. Assume the signal arrives from 54 degrees azimuth and five degrees elevation. Estimate the arrival angle, and then plot the correlation function versus lag for a pair of elements.

Load the signal and extract a small portion for computation.

```
load gong;
y1 = y(1:100);
```

Set up the receiving array.

```
N = 5;
d = 0.25;
sMic = phased.OmnidirectionalMicrophoneElement;
sURA = phased.URA([N,N],[d,d],'Element',sMic);
```

Simulate the arriving plane wave using the `WidebandCollector` System object™.

```
c = 340.0;
arrivalAng = [54;5];
sWBC = phased.WidebandCollector('Sensor',sURA,...
    'PropagationSpeed',c,...
    'SampleRate',Fs,...
    'ModulatedInput',false);
signal = step(sWBC,y1,arrivalAng);
```

Estimate direction of arrival. Choose 10 sensors to correlate with the first element of the URA.

```
sensorpairs = [[2,4,6,8,10,12,14,16,18,20];ones(1,10)];
sGCC = phased.GCCEstimator('SensorArray',sURA,...
    'PropagationSpeed',c,'SampleRate',Fs,...
    'SensorPairSource','Property',...
    'SensorPair',sensorpairs,...
    'DelayOutputPort',true,'CorrelationOutputPort',true);
[estimatedAng,taus,R,lags] = step(sGCC,signal);
```

The estimated angle is:

```
disp(estimatedAng)
```

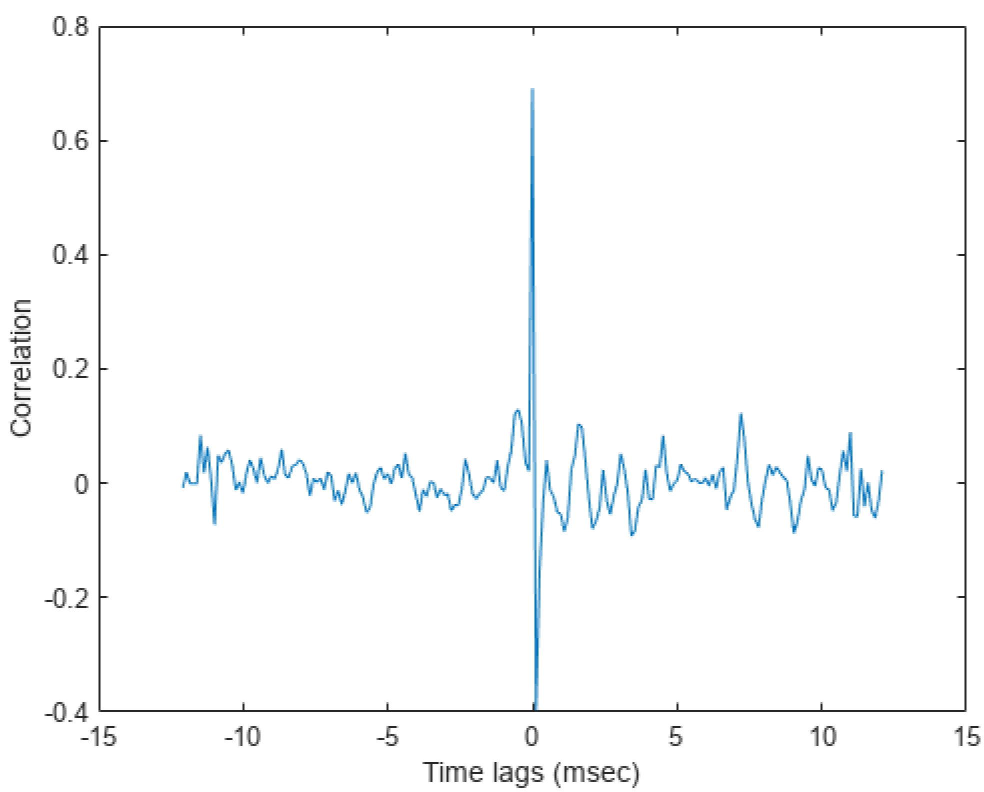
```
   11.6720
    4.2189
```

Plot the correlation between sensor 8 and sensor 1. This pair corresponds to the fourth column of the correlation matrix. The estimated value of tau (in milliseconds) for this pair is:

```
disp(1000*taus(4))
```

```
    0.0238
```

```
plot(1000*lags,real(R(:,4)))
xlabel('Time lags (msec)')
ylabel('Correlation')
```



Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2015b

References

[1] Charles H. Knapp and Carter, G.C., *The Generalized Correlation Method for Estimation of Time Delay*, IEEE Transactions on Acoustics, Speech and Signal Processing, Vol, ASSP-24, No. 4. August 1976.

[2] G. Clifford Carter *Coherence and Time Delay Estimation*, Proceedings of the IEEE, vol 75, No 2, Feb 1987.

See Also

`phased.BeamscanEstimator` | `phased.RootMUSICEstimator`

phased.GSCBeamformer

Package: phased

Generalized sidelobe canceler beamformer

Description

The phased.GSCBeamformer System object implements a generalized sidelobe cancellation (GSC) beamformer. A GSC beamformer splits the incoming signals into two channels. One channel goes through a conventional beamformer path and the second goes into a sidelobe canceling path. The algorithm first pre-steers the array to the beamforming direction and then adaptively chooses filter weights to minimize power at the output of the sidelobe canceling path. The algorithm uses least mean squares (LMS) to compute the adaptive weights. The final beamformed signal is the difference between the outputs of the two paths.

To compute the beamformed signal:

1. Create the phased.GSCBeamformer object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
beamformer = phased.GSCBeamformer
beamformer = phased.GSCBeamformer(Name,Value)
```

Description

beamformer = phased.GSCBeamformer creates a GSC beamformer System object, beamformer, with default property values.

beamformer = phased.GSCBeamformer(Name,Value) creates a GSC beamformer object, beamformer, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN). Enclose each property name in single quotes.

Example: beamformer = phased.GSCBeamformer('SensorArray',phased.ULA('NumElements',20),'SampleRate',300e3) sets the sensor array to a uniform linear array (ULA) with default ULA property values except for the number of elements. The beamformer has a sample rate of 300 kHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SensorArray` — Sensor array
phased.ULA array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object. The array cannot contain subarrays.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`.

Example: `3e8`

Data Types: `single` | `double`

`SampleRate` — Sample rate of signal
`1e6` (default) | positive scalar

Sample rate of signal, specified as a positive scalar. Units are in Hz. The System object uses this quantity to calculate the propagation delay in units of samples.

Example: `1e6`

Data Types: `single` | `double`

`FilterLength` — FIR filter length
`1` (default) | positive integer

Length of the signal path FIR filters, specified as a positive integer. This property determines the adaptive filter size for the sidelobe canceling path. The FIR filter for the conventional beamforming path is a delta function of the same length.

Example: `4`

Data Types: `double` | `single`

`LMSStepSize` — Adaptive filter step size factor
`0.1` (default) | positive real-valued scalar

The adaptive filter step size factor, specified as a positive real-valued scalar. This quantity, when divided by the total power in the sidelobe canceling path, sets the actual adaptive filter step size that is used in the LMS algorithm.

Data Types: `double` | `single`

`DirectionSource` — Source of beamforming direction
`'Property'` (default) | `'Input port'`

Source of beamforming direction, specified as `'Property'` or `'Input port'`. Specify whether the beamforming direction comes from the `Direction` property of this object or from the input argument, `ANG`. Values of this property are:

`'Property'`	Specify the beamforming direction using the `Direction` property.
`'Input port'`	Specify the beamforming direction using the input argument, `ANG`.

Data Types: `char`

`Direction` — Beamforming directions
`[0;0]` (default) | real-valued 2-by-1 vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 vector or a real-valued 2-by-*L* matrix. For a matrix, each column specifies a different beamforming direction. Each column has the form `[AzimuthAngle;ElevationAngle]`. Azimuth angles must lie between –180° and 180° and elevation angles must lie between –90° and 90°. All angles are defined with respect to the local coordinate system of the array. Units are in degrees.

Example: `[40;30]`

Dependencies

To enable this property, set the `DirectionSource` property to `'Property'`.

Data Types: `single` | `double`

Usage

Syntax

```
Y = beamformer(X)
Y = beamformer(X,ANG)
```

Description

`Y = beamformer(X)` performs GSC beamforming on the input, `X`, and returns the beamformed output, `Y`.

`Y = beamformer(X,ANG)` uses `ANG` as the beamforming direction. To use this syntax, set the DirectionSource property to `'Input port'`.

Input Arguments

`X` — Input signal
complex-valued *M*-by-*N* matrix

Input signal, specified as a complex-valued *M*-by-*N* matrix. *M* is the signal length and *N* is the number of array elements specified in the SensorArray property. *M* must be larger than the length of the filter specified by the FilterLength property.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double` | `single`
Complex Number Support: Yes

`ANG` — Beamforming directions
`[0;0]` (default) | real-valued 2-by-1 column vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 column vector The vector has the form `[AzimuthAngle;ElevationAngle]`. Units are in degrees. The azimuth angle must lie between –180° and 180°, and the elevation angle must lie between –90° and 90°.

Example: `[40;10]`

Dependencies

To enable this argument, set the DirectionSource property to `'Input port'`.

Data Types: `double`

Output Arguments

`Y` — Beamformed output
complex-valued 1-by-*M* vector

Beamformed output, returned as a complex-valued 1-by-*M*vector, where *M* is the number of rows of the input X.

Data Types: `double` | `single`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Generalized Sidelobe Cancellation on Uniform Linear Array

Create a GSC beamformer for a 11-element acoustic array in air. A chirp signal is incident on the array at -50° in azimuth and 0° in elevation. Compare the GSC beamformed signal to a Frost beamformed signal. The signal propagation speed is 340 m/s and the sample rate is 8 kHz.

Create the microphone and array System objects. The array element spacing is one-half wavelength. Set the signal frequency to the one-half the Nyquist frequency.

```
c = 340.0;
fs = 8.0e3;
fc = fs/2;
lam = c/fc;
transducer = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[20 20000]);
array = phased.ULA('Element',transducer,'NumElements',11,'ElementSpacing',lam/2);
```

Simulate a chirp signal with a 500 Hz bandwidth.

```
t = 0:1/fs:.5;
signal = chirp(t,0,0.5,500);
```

Create an incident wave arriving at the array. Add gaussian noise to the wave.

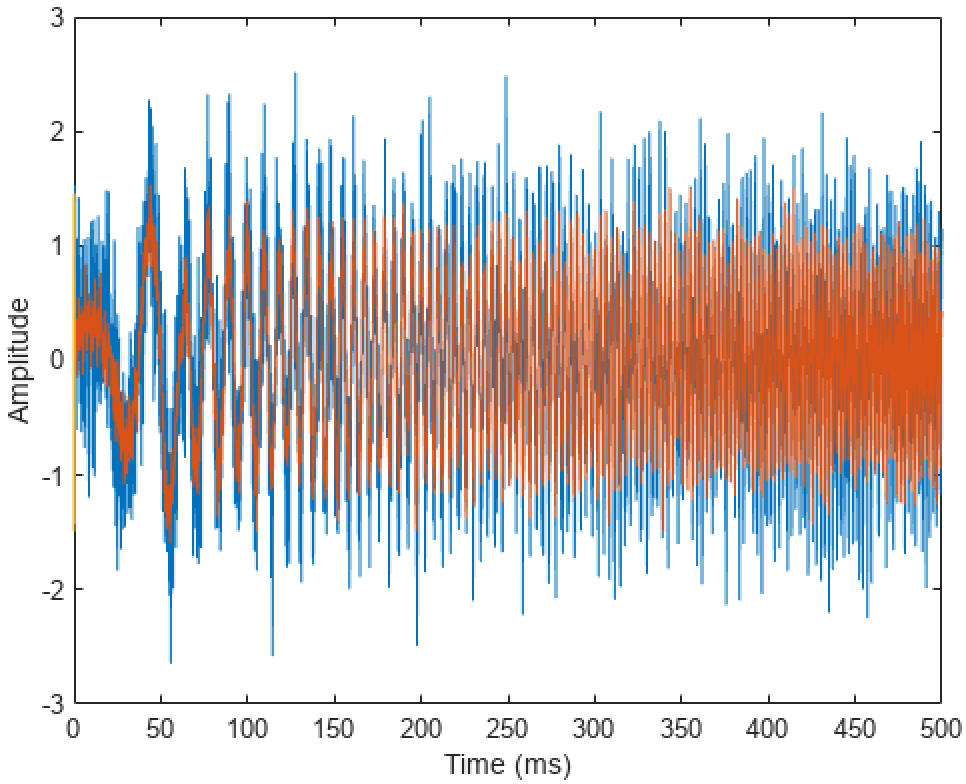
```
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c, ...
    'SampleRate',fs,'ModulatedInput',false,'NumSubbands',512);
incidentAngle = [-50;0];
signal = collector(signal.',incidentAngle);
noise = 0.5*randn(size(signal));
recsignal = signal + noise;
```

Perform Frost beamforming at the actual incident angle.

```
frostbeamformer = phased.FrostBeamformer('SensorArray',array,'PropagationSpeed', ...
    c,'SampleRate',fs,'Direction',incidentAngle,'FilterLength',15);
yfrost = frostbeamformer(recsignal);
```

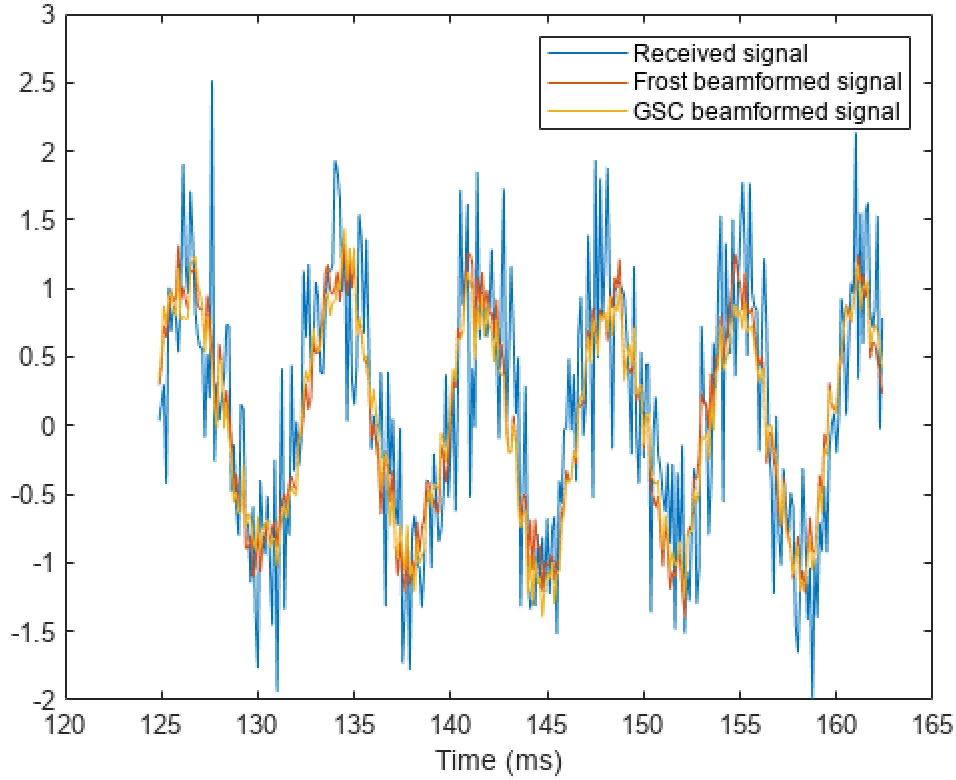
Perform GSC beamforming and plot the beamformer output against the Frost beamformer output. Also plot the nonbeamformed signal arriving at the middle element of the array.

```
gscbeamformer = phased.GSCBeamformer('SensorArray',array, ...
    'PropagationSpeed',c,'SampleRate',fs,'Direction',incidentAngle, ...
    'FilterLength',15);
ygsc = gscbeamformer(recsignal);
plot(t*1000,recsignal(:,6),t*1000,yfrost,t,ygsc)
xlabel('Time (ms)')
ylabel('Amplitude')
```



Zoom in on a small portion of the output.

```
idx = 1000:1300;
plot(t(idx)*1000,recsignal(idx,6),t(idx)*1000,yfrost(idx),t(idx)*1000,ygsc(idx))
xlabel('Time (ms)')
legend('Received signal','Frost beamformed signal','GSC beamformed signal')
```



Generalized Sidelobe Cancellation in Two Directions

Create a GSC beamformer for an 11-element acoustic array in air. A chirp signal is incident on the array at -50° in azimuth and 0° in elevation. Compute the beamformed signal in the direction of the incident wave and in another direction. Compare the two beamformed outputs. The signal propagation speed is 340 m/s and the sample rate is 8 kHz. Create the microphone and array System objects. The array element spacing is one-half wavelength. Set the signal frequency to the one-half the Nyquist frequency.

```
c = 340.0;
fs = 8.0e3;
fc = fs/2;
lam = c/fc;
transducer = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[20 20000]);
array = phased.ULA('Element',transducer,'NumElements',11,'ElementSpacing',lam/2);
```

Simulate a chirp signal with a 500 Hz bandwidth.

```
t = 0:1/fs:0.5;
signal = chirp(t,0,0.5,500);
```

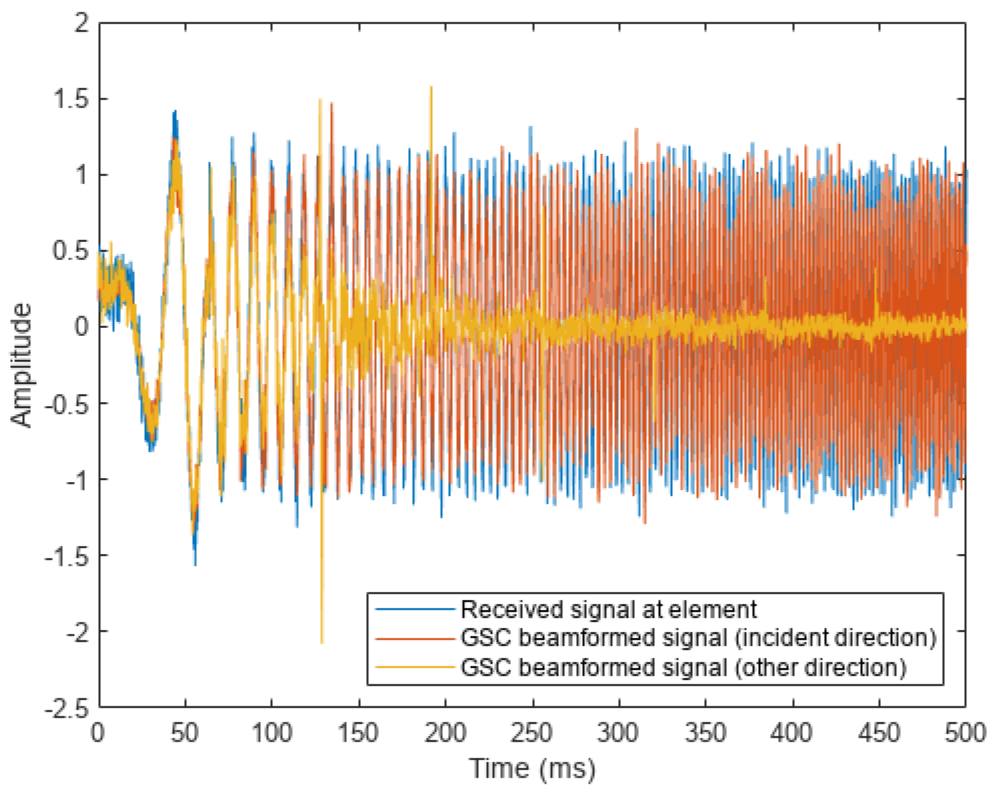
Create an incident wavefield hitting the array.

```
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c, ...
    'SampleRate',fs,'ModulatedInput',false,'NumSubbands',512);
```

```
incidentAngle = [-50;0];
signal = collector(signal.',incidentAngle);
noise = 0.1*randn(size(signal));
recsignal = signal + noise;
```

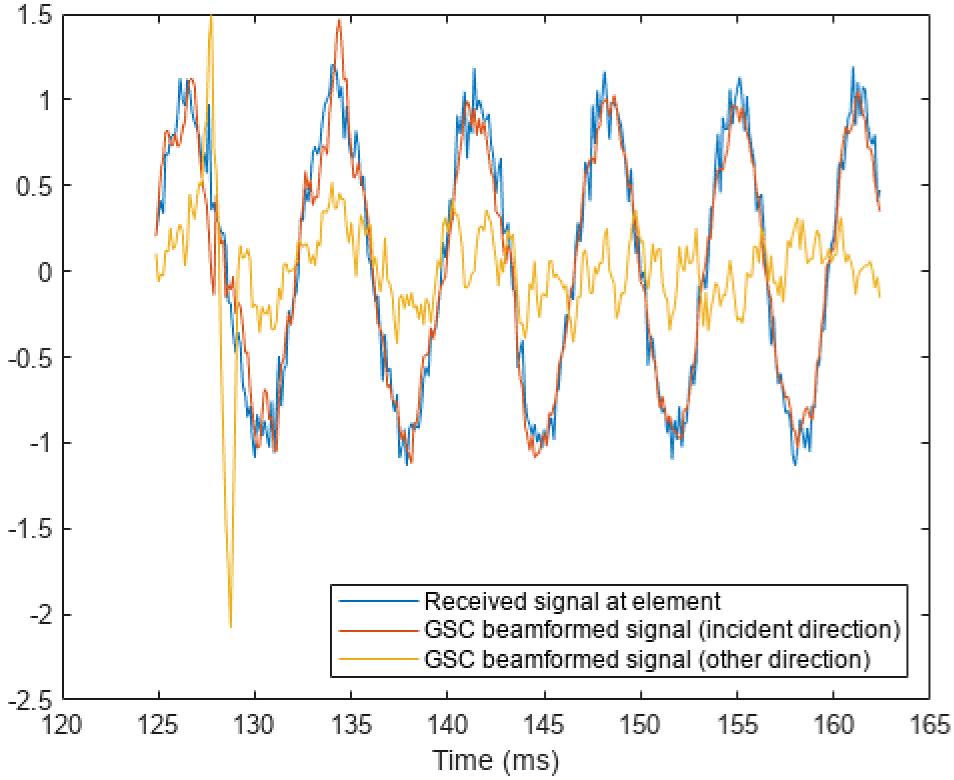
Perform GSC beamforming and plot the beamformer outputs. Also plot the nonbeamformed signal arriving at the middle element of the array.

```
gscbeamformer = phased.GSCBeamformer('SensorArray',array, ...
    'PropagationSpeed',c,'SampleRate',fs,'DirectionSource','Input port', ...
    'FilterLength',5);
ygsci = gscbeamformer(recsignal,incidentAngle);
ygsco = gscbeamformer(recsignal,[20;30]);
plot(t*1000,recsignal(:,6),t*1000,ygsci,t*1000,ygsco)
xlabel('Time (ms)')
ylabel('Amplitude')
legend('Received signal at element','GSC beamformed signal (incident direction)', ...
    'GSC beamformed signal (other direction)','Location','southeast')
```



Zoom in on a small portion of the output.

```
idx = 1000:1300;
plot(t(idx)*1000,recsignal(idx,6),t(idx)*1000,ygsci(idx),t(idx)*1000,ygsco(idx))
xlabel('Time (ms)')
legend('Received signal at element','GSC beamformed signal (incident direction)', ...
    'GSC beamformed signal (other direction)','Location','southeast')
```



Algorithms

Generalized Sidelobe Cancellation

The generalized sidelobe canceler (GSC) is an efficient implementation of a linear constraint minimum variance (LCMV) beamformer. LCMV beamforming minimizes the output power of an array while preserving the power in one or more specified directions. This type of beamformer is called a *constrained beamformer*. You can compute exact weights for the constrained beamformer but the computation is costly when the number of elements is large. The computation requires the inversion of a large spatial covariance matrix. The GSC formulation converts the adaptive constrained optimization LCMV problem into an adaptive unconstrained problem, which simplifies the implementation.

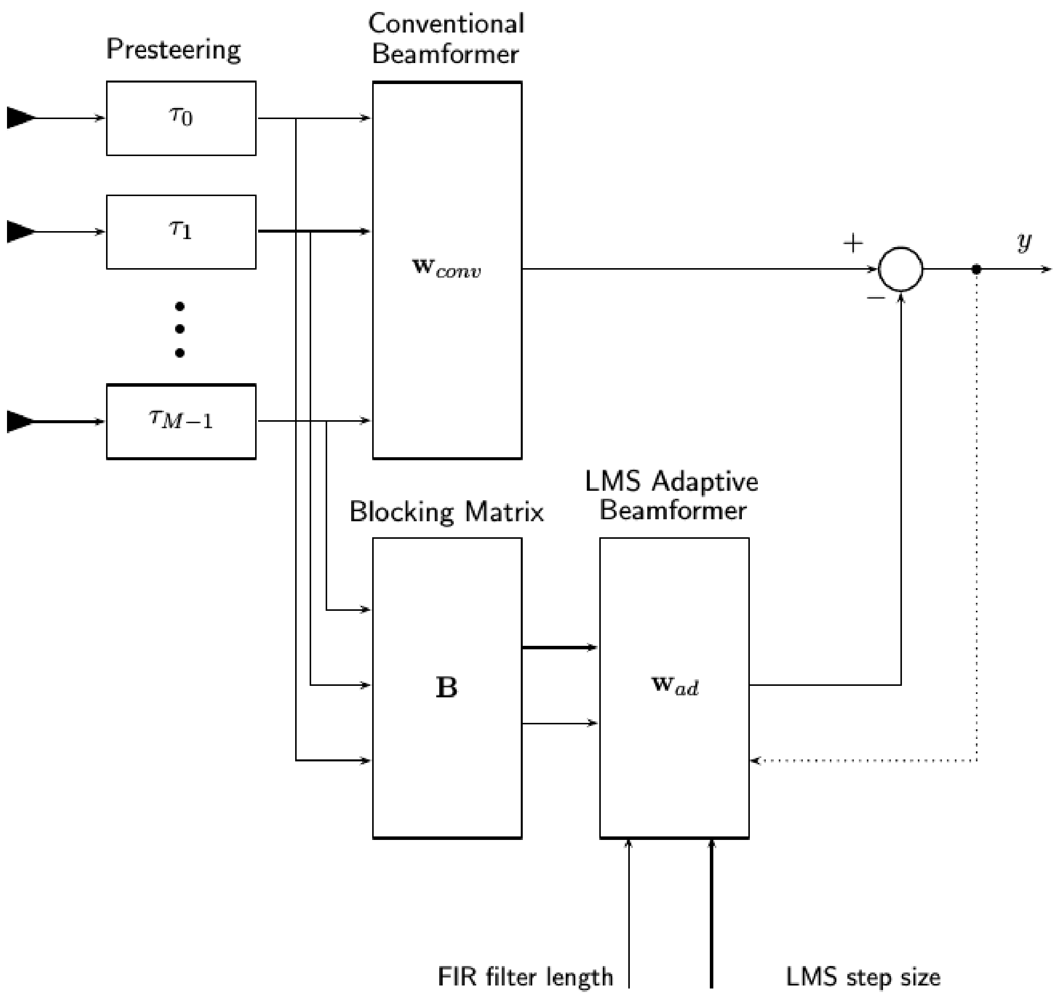
In the GSC algorithm, incoming sensor data is split into two signal paths as shown in the block diagram. The upper path is a conventional beamformer. The lower path is an adaptive unconstrained beamformer whose purpose is to minimize the GSC output power. The GSC algorithm consists of these steps:

1. Presteer the element sensor data by time-shifting the incoming signals. Presteering time-aligns all sensor element signals. The time shifts depend on the arrival angle of the signal.
2. Pass the presteered signals through the upper path into a conventional beamformer with fixed weights, $\mathbf{w}_{conv}$.
3. Also pass the presteered signals through the lower path into the blocking matrix, $\mathbf{B}$. The blocking matrix is orthogonal to the signal and removes the signal from the lower path.

4 Filter the lower path signals through a bank of FIR filters. The FilterLength property sets the length of the filters. The filter coefficients are the adaptive filter weights, $\mathbf{w}_{ad}$.

5 Compute the difference between the upper and lower signal paths. This difference is the beamformed GSC output.

6 Feed the beamformed output back into the filter. The filter adapts its weights using a least mean-square (LMS) algorithm. The actual adaptive LMS step size is equal to the value of the LMSStepSize property divided by the total signal power.



For more information, see [1].

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2016b

References

[1] Griffiths, L. J., and Charles W. Jim. "An alternative approach to linearly constrained adaptive beamforming." *IEEE Transactions on Antennas and Propagation*, 30.1 (1982): 27-34.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

[3] Johnson, D.H., and Dan E. Dudgeon, *Array Signal Processing*, Englewood Cliffs: Prentice-Hall, 1993.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Requires dynamic memory allocation. See “Limitations for System Objects that Require Dynamic Memory Allocation”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.FrostBeamformer` | `phased.MVDRBeamformer` | `phased.PhaseShiftBeamformer` | `phased.SubbandPhaseShiftBeamformer` | `phased.TimeDelayBeamformer` | `phased.TimeDelayLCMVBeamformer`

phased.HeterogeneousConformalArray

Package: `phased`

Heterogeneous conformal array

Description

The `HeterogeneousConformalArray` object constructs a conformal array from a heterogeneous set of antenna elements. A heterogeneous array is an array which consists of different kinds of antenna elements or an array of different kinds of microphone elements. A conformal array can have elements in any position pointing in any direction.

To compute the response for each element in the array for specified directions:

1. Define and set up your conformal array. See "Construction" on page 1-491.
2. Call `step` to compute the response according to the properties of `phased.HeterogeneousConformalArray`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.HeterogeneousConformalArray` creates a heterogeneous conformal array System object, `H`. This object models a heterogeneous conformal array formed with different kinds of sensor elements.

`H = phased.HeterogeneousConformalArray(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`ElementSet`

Set of elements used in the array

Set of elements used in the sensor array, specified as a row MATLAB cell array. Elements specified in the `ElementSet` property must be either Phased Array System Toolbox antennas, microphones, or transducers System objects or Antenna Toolbox System object. In addition, all specified antenna elements must have the same polarization capability.

Default: One cell containing one isotropic antenna element

`ElementIndices`

Elements location assignment

This property specifies the mapping of elements in the array. The property assigns elements to their locations in the array using the indices into the `ElementSet` property. The value of `ElementIndices` must be an length-*N* row vector. In this vector, *N* represents the number of elements in the array. The values in the vector specified by `ElementIndices` must be less than or equal to the number of entries in the `ElementSet` property.

Default: `[1 2 2 1]`

`ElementPosition`

Element positions

`ElementPosition` specifies the positions of the elements in the conformal array. The value of the `ElementPosition` property must be a 3-by-*N* matrix, where *N* indicates the number of elements in the conformal array. Each column of `ElementPosition` represents the position, in the form `[x; y; z]` (in meters), of a single element in the local coordinate system of the array. The local coordinate system has its origin at an arbitrary point.

Default: `[0; 0; 0]`

`ElementNormal`

Element normal directions

`ElementNormal` specifies the normal directions of the elements in the conformal array. Angle units are degrees. The value assigned to `ElementNormal` must be either a 2-by-*N* matrix or a 2-by-1 column vector. The variable *N* indicates the number of elements in the array. If the value of `ElementNormal` is a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the value of `ElementNormal` is a 2-by-1 column vector, it specifies the pointing direction of all elements in the array.

You can use the `ElementPosition` and `ElementNormal` properties to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Default: `[0; 0]`

`Taper`

Element taper or weighting

Element tapering or weighting, specified as a complex-valued scalar, 1-by-*N* row vector, or *N*-by-1 column vector. The quantity *N* is the number of elements in the array as determined by the size of the `ElementIndices` property. Tapers, also known as weights, are applied to each sensor element in the sensor array and modify both the amplitude and phase of the received data. If `'Taper'` is a scalar, the same taper value is applied to all elements. If `'Taper'` is a vector, each taper value is applied to the corresponding sensor element.

Default: 1

Methods

Specific to `phased.HeterogeneousConformalArray` Object	
`beamwidth`	Compute and display beamwidth of an array
`collectPlaneWave`	Simulate received plane waves
`directivity`	Directivity of heterogeneous conformal array
`getElementNormal`	Normal vector to array elements
`getElementPosition`	Positions of array elements
`getNumElements`	Number of elements in array
`getTaper`	Array element tapers
`isPolarizationCapable`	Polarization capability
`pattern`	Plot heterogeneous conformal array pattern
`patternAzimuth`	Plot heterogeneous conformal array directivity or pattern versus azimuth
`patternElevation`	Plot heterogeneous conformal array directivity or pattern versus elevation
`perturbations`	Perturbations defined on phased array
`perturbedArray`	Apply perturbations to phased array
`perturbedPattern`	Compute and plot azimuth pattern of perturbed array
`step`	Output responses of array elements
`viewArray`	View array geometry

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Heterogeneous Uniform Circular Array

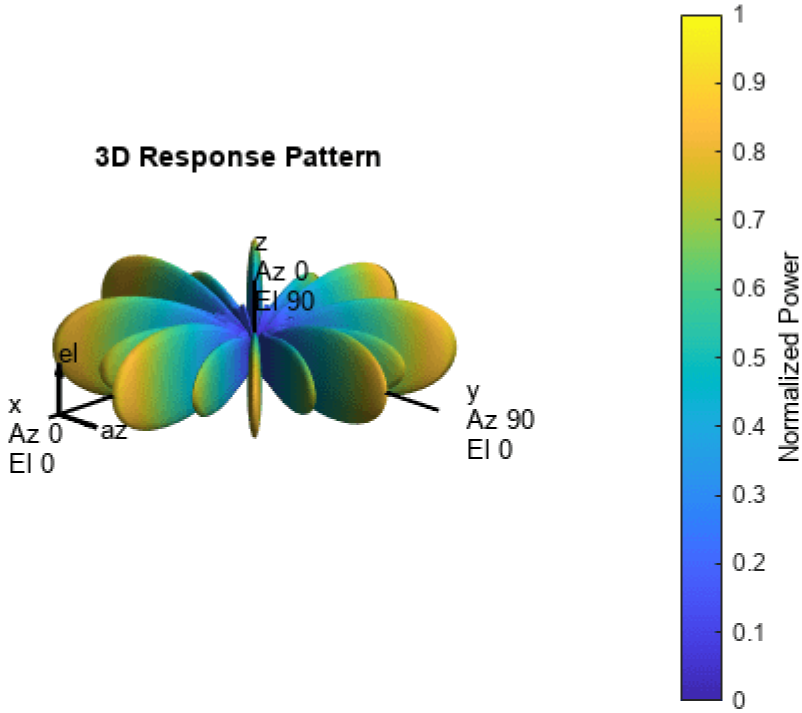
Construct an 8-element heterogeneous uniform circular array (UCA) using the `ConformalArray` System object™. Four of the elements have a cosine pattern with a power of 1.6 while the remaining elements have a cosine pattern with a power of 2.0. Plot the 3-D power response. Assume a 1 GHz operating frequency. The wave propagation speed is the speed of light.

Construct the array

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.6);
sElement2 = phased.CosineAntennaElement('CosinePower',2.0);
N = 8;
azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2],...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
c = physconst('LightSpeed');
fc = 1e9;
```

Create the 3-D power pattern

```
pattern(sArray,fc,[-180:180],[-90:90],...
    'CoordinateSystem','polar',...
    'Type','power')
```



Version History

Introduced in R2013a

References

[1] Josefsson, L. and P. Persson. *Conformal Array Antenna Theory and Design*. Piscataway, NJ: IEEE Press, 2006.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, `plotResponse`, and `viewArray` methods are not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.UCA` | `phased.URA` | `phased.HeterogeneousULA` | `phased.HeterogeneousURA` | `phased.ConformalArray` | `uv2azel` | `phitheta2azel`

Topics

"Phased Array Gallery"

directivity

System object: `phased.HeterogeneousConformalArray`
Package: `phased`

Directivity of heterogeneous conformal array

Syntax

```
D = directivity(H,FREQ,ANGLE)
D = directivity(H,FREQ,ANGLE,Name,Value)
```

Description

`D = directivity(H,FREQ,ANGLE)` computes the "Directivity" on page 1-499 of a heterogeneous conformal array of antenna or microphone elements, `H`, at frequencies specified by the `FREQ` and in angles of direction specified by the `ANGLE`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`D = directivity(H,FREQ,ANGLE,Name,Value)` computes the directivity with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`H` — Heterogeneous conformal array

System object

Heterogeneous conformal array specified as a `phased.HeterogeneousConformalArray` System object.

Example: `H = phased.HeterogeneousConformalArray;`

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`ANGLE` — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PropagationSpeed` — Signal propagation speed

speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights

1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Heterogeneous Conformal Array

Compute the directivity of a steered heterogeneous conformal array. Construct a 24-element heterogeneous disk array using elements having different antenna patterns and then show how to compute the array's directivity.

Set the signal speed to the speed of light and the signal frequency to 2GHz.

```
c = physconst('LightSpeed');
freq = 2e9;
```

Choose two different types of elements - both are cosine antenna elements with different powers.

```
myElement1 = phased.CosineAntennaElement('CosinePower',1.5);
myElement2 = phased.CosineAntennaElement('CosinePower',1.8);
```

Set up a three-ring disk array with 8 elements per ring. The inner ring has different elements from the outer rings.

```
N = 8;
azang = (0:N-1)*360/N-180;
p0 = [zeros(1,N);cosd(azang);sind(azang)];
posn = [0.6*p0, 0.4*p0, 0.2*p0];
myArray = phased.HeterogeneousConformalArray;
myArray.ElementPosition = posn;
myArray.ElementNormal = zeros(2,3*N);
myArray.ElementSet = {myElement1,myElement2};
myArray.ElementIndices = [1 1 1 1 1 1 1 1,...
    1 1 1 1 1 1 1 1,...
    2 2 2 2 2 2 2 2];
```

Set up the steering vector to point at 30 degrees azimuth and compute the directivity in that direction.

```
lambda = c/freq;
ang = [30;0];
w = steervec(getElementPosition(myArray)/lambda,ang);
d = directivity(myArray,freq,ang,'PropagationSpeed',c,...
            'Weights',w)
```

```
d = 20.9519
```

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternAzimuth` | `patternElevation`

collectPlaneWave

System object: phased.HeterogeneousConformalArray
Package: phased

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

`Y = collectPlaneWave(H,X,ANG)` returns the received signals at the sensor array, `H`, when the input signals indicated by `X` arrive at the array from the directions specified in `ANG`.

`Y = collectPlaneWave(H,X,ANG,FREQ)`, in addition, specifies the incoming signal carrier frequency in `FREQ`.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)`, in addition, specifies the signal propagation speed in `C`.

Input Arguments

H

Array object.

X

Incoming signals, specified as an M-column matrix. Each column of `X` represents an individual incoming signal.

ANG

Directions from which incoming signals arrive, in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column specifies the direction of arrival of the corresponding signal in `X`. Each column of `ANG` is in the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `ANG` is a row vector of length M, each entry in `ANG` specifies the azimuth angle. In this case, the corresponding elevation angle is assumed to be 0°.

FREQ

Carrier frequency of signal in hertz. `FREQ` must be a scalar.

Default: `3e8`

c

Propagation speed of signal in meters per second.

Default: Speed of light

Output Arguments

Y

Received signals. Y is an N-column matrix, where N is the number of elements in the array H. Each column of Y is the received signal at the corresponding array element, with all incoming signals combined.

Examples

Simulate Received Signal at Heterogeneous Conformal Array

Simulate the received signal at an 8-element heterogeneous uniform circular array created using the phased.HeterogeneousConformalArray System object™. The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light.

```
antenna1 = phased.CosineAntennaElement('CosinePower',1.5);
antenna2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8;
azang = (0:N-1)*360/N-180;
array = phased.HeterogeneousConformalArray('ElementPosition', ...
    [cosd(azang);sind(azang);zeros(1,N)],'ElementNormal',[azang;zeros(1,N)], ...
    'ElementSet',{antenna1,antenna2},'ElementIndices',[1 1 1 1 2 2 2 2]);
c = physconst('LightSpeed');
y = collectPlaneWave(array,randn(4,2),[10 30],c);
disp(y(:,1:2))
```

```
   0.7476 + 0.2890i   0.5378 + 0.5554i
   0.9544 - 0.8005i  -0.5059 + 1.3857i
  -2.5374 - 0.5387i  -1.3746 - 2.1411i
   1.0865 + 0.3377i   0.6977 + 0.8549i
```

Algorithms

collectPlaneWave modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. The method does not account for the response of individual elements in the array.

For further details, see Van Trees [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

getElementNormal

System object: `phased.HeterogeneousConformalArray`
Package: `phased`

Normal vector to array elements

Syntax

```
normvec = getElementNormal(sConfArray)
normvec = getElementNormal(sConfArray,elemidx)
```

Description

`normvec = getElementNormal(sConfArray)` returns the normal vectors of the array elements of the `phased.sConfArray` System object, `sConfArray`. The output argument `normvec` is a 2-by-*N* matrix, where *N* is the number of elements in array, `sConfArray`. Each column of `normvec` defines the normal direction of an element in the local coordinate system in the form`[az;el]`. Units are degrees. The origin of the local coordinate system is defined by the phase center of the array.

`normvec = getElementNormal(sConfArray,elemidx)` returns only the normal vectors of the elements specified in the element index vector, `elemidx`. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

`sConfArray` — Heterogeneous conformal array
`phased.HeterogeneousConformalArray` System object

Heterogeneous conformal array, specified as a `phased.HeterogeneousConformalArray` System object.

Example: `phased.HeterogeneousConformalArray`

`elemidx` — Element indices
all array elements (default) | integer-valued 1-by-*M* row vector | integer-valued *M*-by-1 column vector

Element indices, specified as a 1-by-*M* or *M*-by-1 vector. Index values lie in the range 1 to *N* where *N* is the number of elements of the array. When `elemidx` is specified, `getElementNormal` returns the normal vectors of the elements contained in `elemidx`.

Example: `[1,5,4]`

Output Arguments

`normvec` — Element normal vectors
2-by-*P* real-valued vector

Element normal vectors, specified as a 2-by-*P* real-valued vector. Each column of `normvec` takes the form `[az,el]`. When `elemidx` is not specified, *P* equals the array dimension. When `elemidx` is specified, *P* equals the length of `elemidx`, *M*.

Examples

Display Heterogeneous Conformal Array Element Normals

Construct a 5-element acoustic cross array (UCA) composed of two different types of cosine antenna elements. Use the `Phased.HeterogeneousConformalArray` System object™. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s. Display the array normal vectors.

```
N = 5;
fc = 4000;
c = 1500.0;
lam = c/fc;
x = zeros(1,N);
y = [-1,0,1,0,0]*lam/2;
z = [0,0,0,-1,1]*lam/2;
sCos1 = phased.CosineAntennaElement('CosinePower',1.5);
sCos2 = phased.CosineAntennaElement('CosinePower',1.8);
sHCA = phased.HeterogeneousConformalArray('ElementSet',{sCos1,sCos2},...
    'ElementIndices',[1,2,2,2,1],...
    'ElementPosition',[x;y;z],...
    'ElementNormal',[[-20,-10,0,10,20];zeros(1,N)]);
pos = getElementPosition(sHCA)
```

```
pos = 3×5

         0         0         0         0         0
   -0.1875         0    0.1875         0         0
         0         0         0   -0.1875    0.1875
```

```
normvec = getElementNormal(sHCA)
```

```
normvec = 2×5

   -20   -10     0    10    20
     0     0     0     0     0
```

Version History

Introduced in R2016a

getElementPosition

System object: phased.HeterogeneousConformalArray
Package: phased

Positions of array elements

Syntax

```
pos = getElementPosition(sHCA)
pos = getElementPosition(sHCA,elemidx)
```

Description

pos = getElementPosition(sHCA) returns the element positions of the HeterogeneousConformalArray System object, sHCA. POS is an 3-by-*N* matrix where *N* is the number of elements in H. Each column of pos defines the position of an element in the local coordinate system, in meters, in the form [x;y;z].

For details regarding the local coordinate system of the conformal or heterogeneous conformal array, enter phased.ConformalArray.coordinateSystemInfo.

pos = getElementPosition(sHCA,elemidx) returns the positions of the elements that are specified in the element index vector elemidx.

Examples

Element Positions of Heterogeneous Conformal Array

Construct an 8-element heterogeneous conformal array with oriented short-dipole antenna elements. Then, obtain the positions of the first four elements.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
N = 8; azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    [cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)],...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
pos = getElementPosition(sArray);
disp(pos(:,1:4));
```

```
   -1.0000   -0.7071         0    0.7071
         0   -0.7071   -1.0000   -0.7071
         0         0         0         0
```

getNumElements

System object: `phased.HeterogeneousConformalArray`
Package: `phased`

Number of elements in array

Syntax

`N = getNumElements(H)`

Description

`N = getNumElements(H)` returns the number of elements, N, in the conformal array object `H`.

Examples

Number of Elements in Heterogeneous Conformal Array

Construct a heterogeneous 8-element circular array and use the `getNumElements` method to return the number of elements.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
N = 8;
azang = (0:N-1)*360/N-180;
array = phased.HeterogeneousConformalArray('ElementPosition', ...
    [cosd(azang);sind(azang);zeros(1,N)], ...
    'ElementNormal',[azang;zeros(1,N)], ...
    'ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
N = getNumElements(array)
```

```
N = 8
```

getTaper

System object: phased.HeterogeneousConformalArray
Package: phased

Array element tapers

Syntax

wts = getTaper(h)

Description

wts = getTaper(h) returns the tapers applied to each element of a conformal array, h. Tapers are often referred to as weights.

Input Arguments

h — Conformal array

phased.HeterogeneousConformalArray System object

Conformal array specified as a phased.HeterogeneousConformalArray System object.

Output Arguments

wts — Array element tapers

N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1, complex-valued vector, where *N* is the number of elements in the array.

Examples

Create Tapered Heterogeneous Conformal Disk Array

Create a 12-element, 2-ring tapered disk array where the outer ring is more heavily tapered than the inner ring.

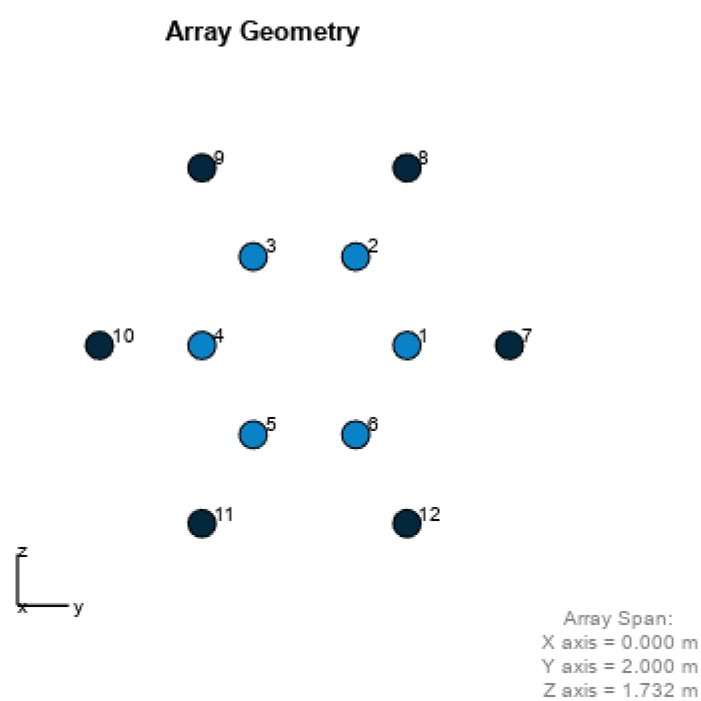
```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
elemAngles = ([0:5]*360/6);
elemPosInner = 0.5*[zeros(size(elemAngles));cosd(elemAngles); ...
    sind(elemAngles)];
elemPosOuter = [zeros(size(elemAngles));cosd(elemAngles); ...
    sind(elemAngles)];
elemNorms = repmat([0;0],1,12);
taper =  [ones(size(elemAngles)),0.3*ones(size(elemAngles))];
```

```
array = phased.HeterogeneousConformalArray('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 1 1 1 1 1 2 2 2 2 2 2], ...
    'ElementPosition',[elemPosInner,elemPosOuter],'ElementNormal',elemNorms, ...
    'Taper',taper);
w = getTaper(array)
```

```
w = 12×1

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    0.3000
    0.3000
    0.3000
    0.3000
      ⋮
```

Draw the array showing taper colors.

```
viewArray(array,'ShowTaper',true,'ShowIndex','all')
```



isPolarizationCapable

System object: phased.HeterogeneousConformalArray
Package: phased

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h — Conformal array

Conformal array specified as a `phased.HeterogeneousConformalArray` System object.

Output Arguments

flag — Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

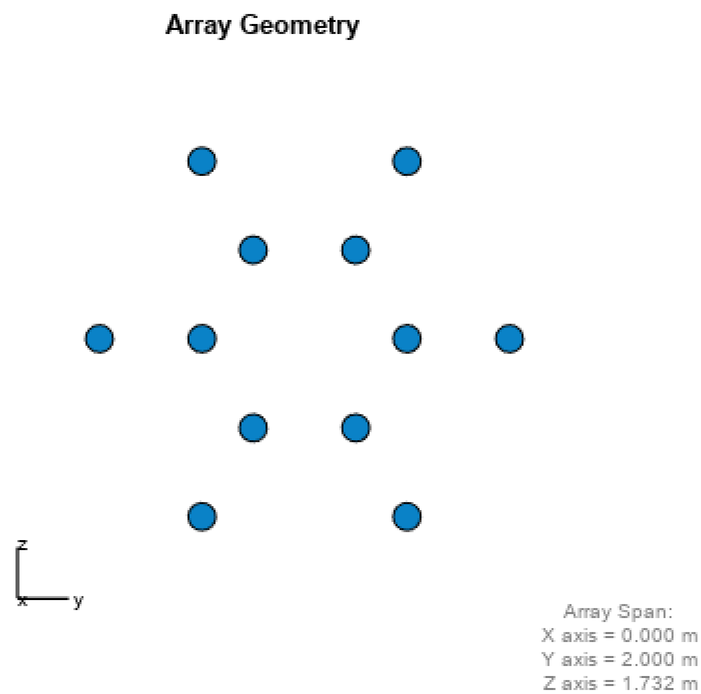
Examples

Conformal Heterogeneous Array Supports Polarization

Show that a disk heterogeneous conformal array of short-dipole antennas supports polarization.

```
antenna1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
elemAngles = ([0:5]*360/6);
elemPosInner = 0.5*[zeros(size(elemAngles));...
    cosd(elemAngles); sind(elemAngles)];
elemPosOuter = [zeros(size(elemAngles));...
    cosd(elemAngles); sind(elemAngles)];
elemNorms = repmat([0;0],1,12);
array = phased.HeterogeneousConformalArray(...
    'ElementSet',{antenna1,antenna2},...
```

```
    'ElementIndices',[1 1 1 1 1 1 2 2 2 2 2 2],...
    'ElementPosition',[elemPosInner,elemPosOuter],...
    'ElementNormal',elemNorms);
viewArray(array)
```




```
isPolarizationCapable(array)
```

```
ans = logical
   1
```

The returned value of 1 shows that this array supports polarization when used with a polarized antenna.

pattern

System object: phased.HeterogeneousConformalArray
Package: phased

Plot heterogeneous conformal array pattern

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sArray,FREQ)` plots the 3-D array directivity pattern (in dBi) for the array specified in `sArray`. The operating frequency is specified in `FREQ`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`pattern(sArray,FREQ,AZ)` plots the array directivity pattern at the specified azimuth angle.

`pattern(sArray,FREQ,AZ,EL)` plots the array directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the array pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See “Convert plotResponse to pattern” on page 1-521 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sArray` — Heterogeneous conformal array

System object

Heterogeneous conformal array, specified as a `phased.HeterogeneousConformalArray` System object.

Example: `sArray= phased.HeterogeneousConformalArray;`

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Orientation` — Array orientation
`[0;0;0]`. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

`Normalize` — Display normalize pattern
`true` (default) | `false`

Display normalized pattern, specified as the comma-separated pair consisting of `'Normalize`' and a Boolean. Set this parameter to `true` to display a normalized pattern. This parameter does not apply when you set `'Type'` to `'directivity'`. Directivity patterns are already normalized.

Data Types: `logical`

`ShowArray` — View array geometry
`false` (default) | `true`

View the array geometry along with the 3D radiation pattern, specified as `false` or `true`.

Data Types: `logical`

`ShowLocalCoordinates` — Show local coordinate axes
`true` (default) | `false`

Show the local coordinate axes, specified as `true` or `false`.

Data Types: `logical`

`ShowColorbar` — Show colorbar
`true` (default) | `false`

Show the colorbar, specified as `true` or `false`.

Data Types: `logical`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights`' and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

PAT — Array pattern

M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of `PAT` correspond to the dimensions of the output arguments `AZ_ANG` and `EL_ANG`.

AZ_ANG — Azimuth angles

scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in `AZ`. The columns of `PAT` correspond to the values in `AZ_ANG`. Units are in degrees.

EL_ANG — Elevation angles

scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

Examples

Plot Power Patterns of 8-Element Heterogeneous Uniform Circular Array

Create an 8-element uniform circular array using the `HeterogeneousConformalArray` System object™ with two different types of short-dipole elements. Then, plot the 3-D and 2-D power patterns.

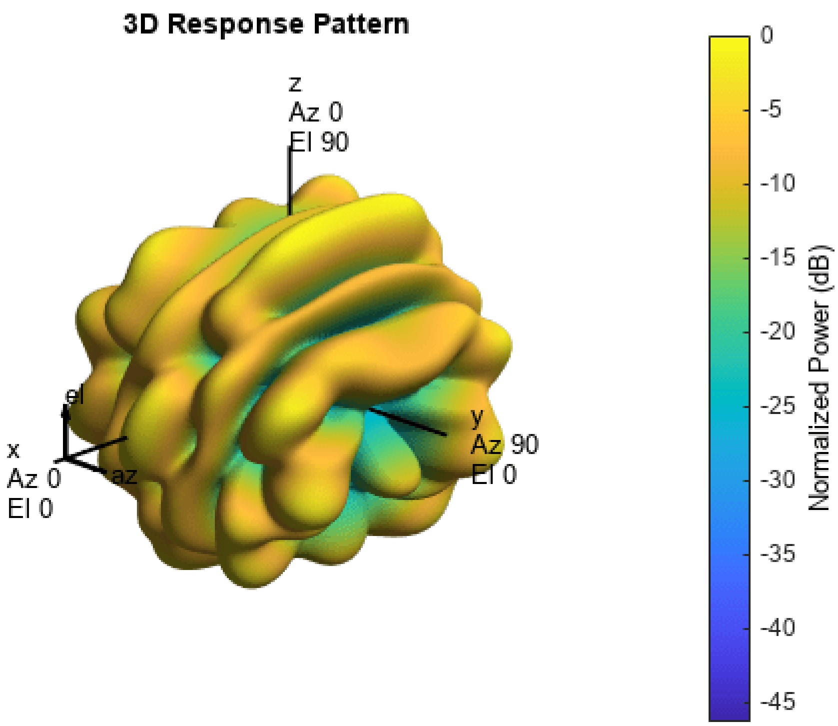
Create the array

```
sElement1 = phased.ShortDipoleAntennaElement('FrequencyRange',[1e9 5e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement('FrequencyRange',[1e9 5e9],...
    'AxisDirection','Y');
N = 8;
azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    0.4*[zeros(1,N);cosd(azang);sind(azang)],...
    'ElementNormal', zeros(2,N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
```

Plot 3-D power pattern

Assume the operating frequency is 1.5 GHz and the wave propagation speed is the speed of light.

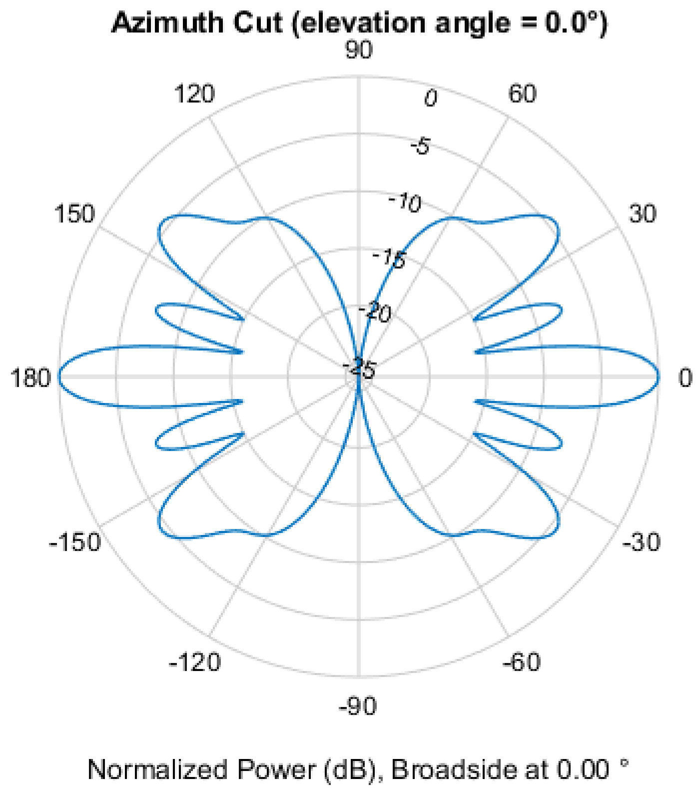
```
c = physconst('LightSpeed');
fc = 1.5e9;
pattern(sArray,fc,[-180:180],[-90:90],...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Polarization','combined')
```



Plot 2-D power pattern

Take a cut of the 3-D power pattern at zero degrees elevation.

```
pattern(sArray,fc,[-180:180],0,...
    'PropagationSpeed',c',...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Polarization','combined')
```



Plot Pattern of Disk Array

Construct a 24-element disk array using elements with two different types of cosine antennas. Then, plot the array pattern.

Create the array

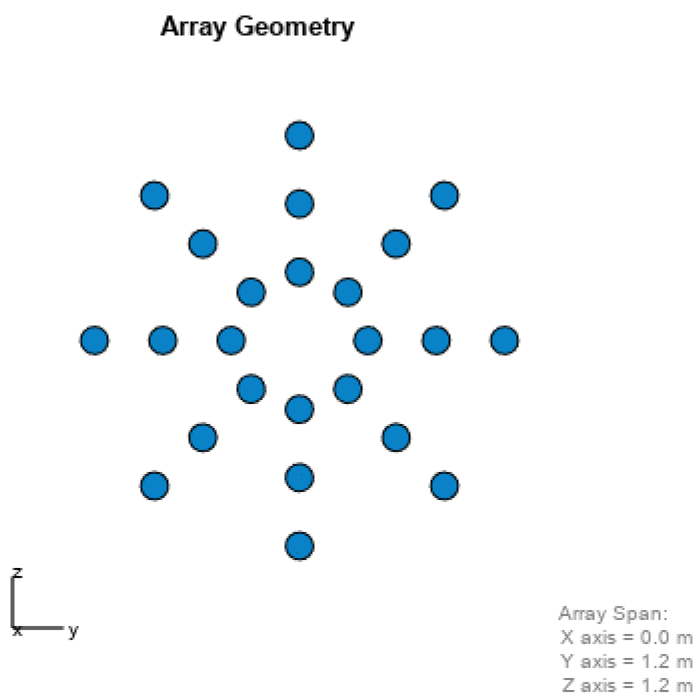
The array consists of cosine antenna elements with different power exponents.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8;
azang = (0:N-1)*360/N-180;
p0 = [zeros(1,N);cosd(azang);sind(azang)];
posn = [0.6*p0, 0.4*p0, 0.2*p0];
sArray1 = phased.HeterogeneousConformalArray(...
```

```
    'ElementPosition',posn,...
    'ElementNormal', zeros(2,3*N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 1 1 1 1,...
    1 1 1 1 1 1 1 1,...
    2 2 2 2 2 2 2 2]);
```

View the disk array

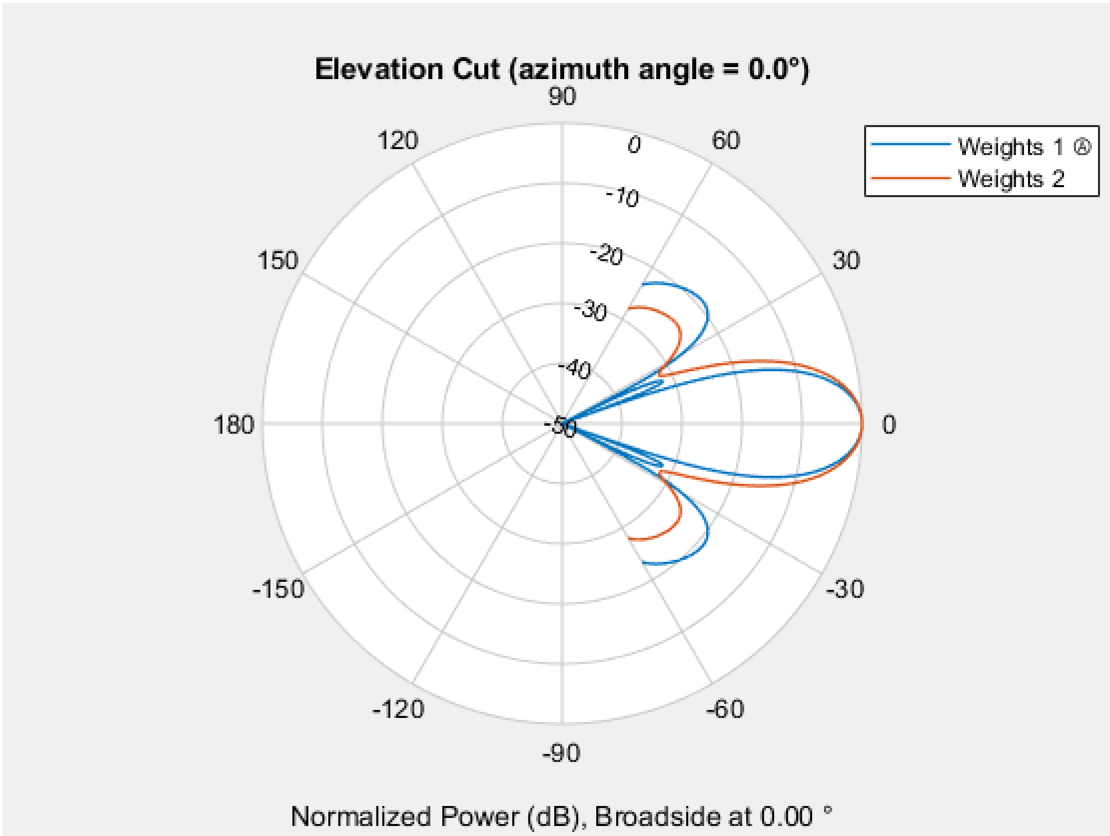
```
viewArray(sArray1)
```



Plot the power pattern

Plot the elevation power pattern of this array two different sets of element weights. The first set is uniform weights on the elements. The second set is a tapered set of weights set by the `Weights` parameter. Restrict the plot of the response from -60 to 60 degrees in 0.1 degree increments. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
c = physconst('LightSpeed');
fc = 1e9;
wts1 = ones(3*N,1);
wts1 = wts1/sum(abs(wts1));
wts2 = [0.5*ones(N,1); 0.7*ones(N,1); 1*ones(N,1)];
wts2 = wts2/sum(abs(wts2));
pattern(sArray1,fc,0,[-60:0.1:60],'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','powerdb','Weights',[wts1,wts2])
```



As expected, the tapered weights broaden the mainlobe and reduce the sidelobes.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on "Element Directivity" and "Array Directivity".

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.

Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.

Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.

'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the UV grid values using AZ and EL.</td>
</tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
	Display space | | 'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.	
'CutAngle' name-value pair	Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.	No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.
'NormalizeResponse' name-value pair	Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.	Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.
'OverlayFreq' name-value pair	Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.	'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot. The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.
'Polarization' name-value pair	Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.	'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.
'Unit' name-value pair	Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.	'Type' name-value pair, uses equivalent options with different names **plotResponse** → **pattern** 'db' → 'powerdb' 'mag' → 'efield' 'pow' → 'power' 'dbi' → 'directivity'
'Weights' name-value pair	Array element tapers (or weights).	'Weights' name-value pair (no change).
'AzimuthAngles' name-value pair	Azimuth angles used to display the antenna or array response.	AZ argument

plotResponse Inputs	plotResponse Description	pattern Inputs
'ElevationAngles' name-value pair	Elevation angles used to display the antenna or array response.	EL argument
'UGrid' name-value pair	Contains *U* coordinates in *UV*-space.	AZ argument when 'CoordinateSystem' name-value pair is set to 'uv'
'VGrid' name-value pair	Contains *V*-coordinates in *UV*-space.	EL argument when 'CoordinateSystem' name-value pair is set to 'uv'

Version History

Introduced in R2015a

See Also

patternAzimuth | patternElevation

patternAzimuth

System object: `phased.HeterogeneousConformalArray`
Package: `phased`

Plot heterogeneous conformal array directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sArray,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(sArray,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sArray,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sArray` — Heterogeneous conformal array

System object

Heterogeneous conformal array, specified as a `phased.HeterogeneousConformalArray` System object.

Example: `sArray= phased.HeterogeneousConformalArray;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

Examples

Plot Azimuthal Directivity Pattern of Disk Array

Construct a 24-element disk array using elements with two different types of cosine antennas. Then, plot the array azimuthal directivity pattern.

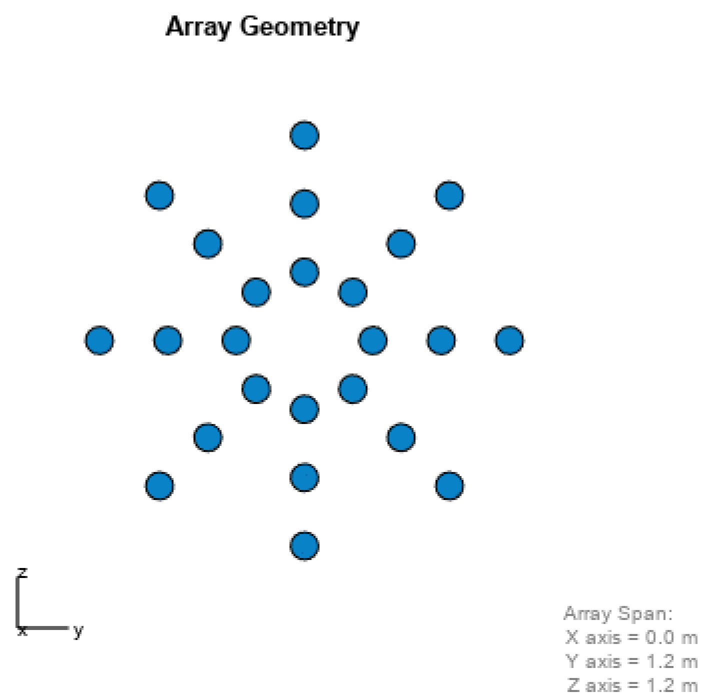
Create the array

The array consists of cosine antenna elements with different power exponents.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8;
azang = (0:N-1)*360/N-180;
p0 = [zeros(1,N);cosd(azang);sind(azang)];
posn = [0.6*p0, 0.4*p0, 0.2*p0];
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',posn,...
    'ElementNormal', zeros(2,3*N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 1 1 1 1,...
    1 1 1 1 1 1 1 1,...
    2 2 2 2 2 2 2 2]);
```

View the disk array

```
viewArray(sArray)
```

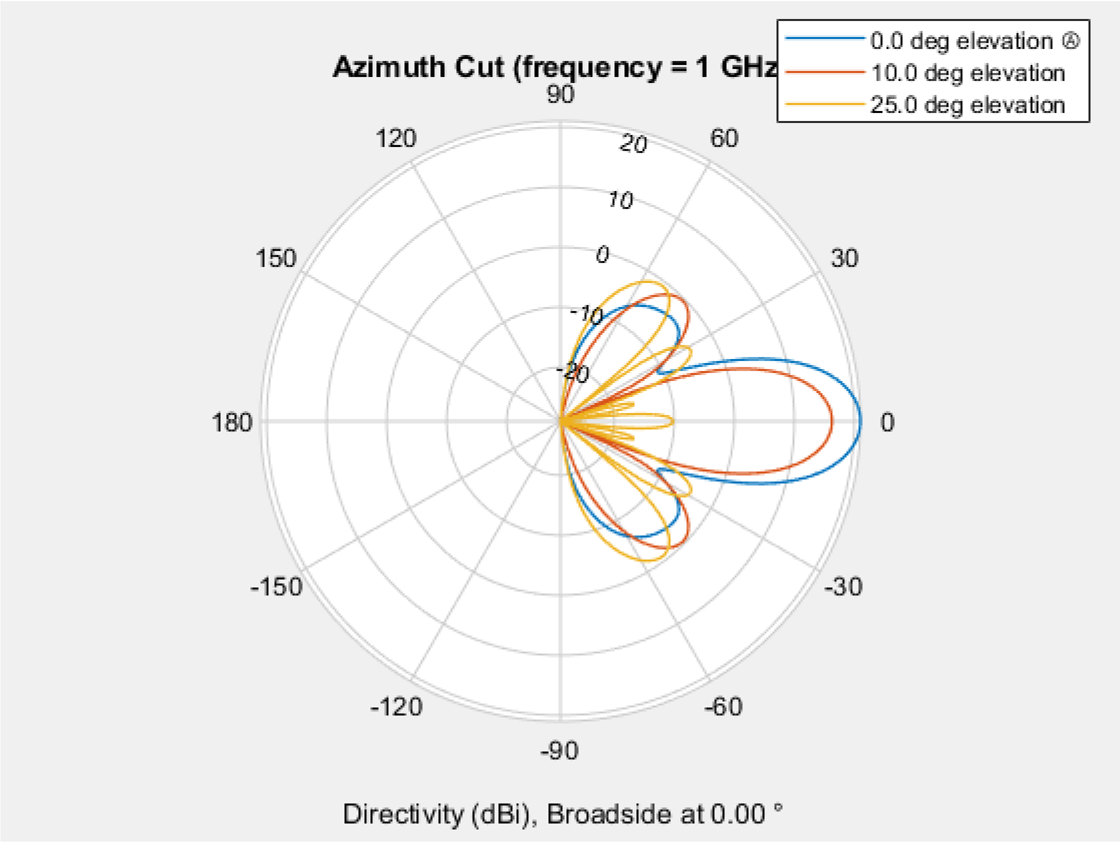


Plot the power pattern

Plot the azimuthal power pattern of this array for three different elevation angles: 0, 10 and 25 degrees. Apply radial tapering to the array. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
c = physconst('LightSpeed');
fc = 1e9;
wts = [0.5*ones(N,1); 0.7*ones(N,1); 1.0*ones(N,1)];
wts = wts/sum(abs(wts));
```

```
patternAzimuth(sArray,fc,[0,10,25],'PropagationSpeed',c,...
    'Type','directivity','Weights',wts)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on "Element Directivity" and "Array Directivity".

Version History

Introduced in R2015a

See Also

`patternElevation` | `pattern`

patternElevation

System object: `phased.HeterogeneousConformalArray`
Package: `phased`

Plot heterogeneous conformal array directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sArray,FREQ)` plots the 2-D array directivity pattern versus elevation (in dBi) for the array `sArray` at zero degrees azimuth angle. When `AZ` is a vector, multiple overlaid plots are created. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternElevation(sArray,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sArray,FREQ,AZ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sArray` — Heterogeneous conformal array

System object

Heterogeneous conformal array, specified as a `phased.HeterogeneousConformalArray` System object.

Example: `sArray= phased.HeterogeneousConformalArray;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

`AZ` — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

`PAT` — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Plot Elevation Directivity Pattern of Disk Array

Construct a 24-element disk array using elements with two different types of cosine antennas. Then, plot the array elevation directivity pattern.

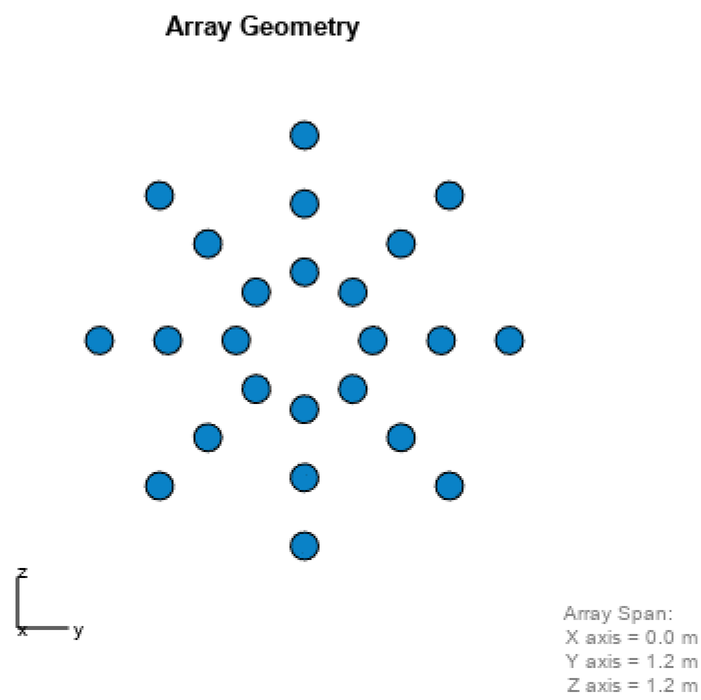
Create the array

The array consists of cosine antenna elements with different power exponents.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8;
azang = (0:N-1)*360/N-180;
p0 = [zeros(1,N);cosd(azang);sind(azang)];
posn = [0.6*p0, 0.4*p0, 0.2*p0];
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',posn,...
    'ElementNormal', zeros(2,3*N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 1 1 1 1,...
    1 1 1 1 1 1 1 1,...
    2 2 2 2 2 2 2 2]);
```

View the disk array

```
viewArray(sArray)
```

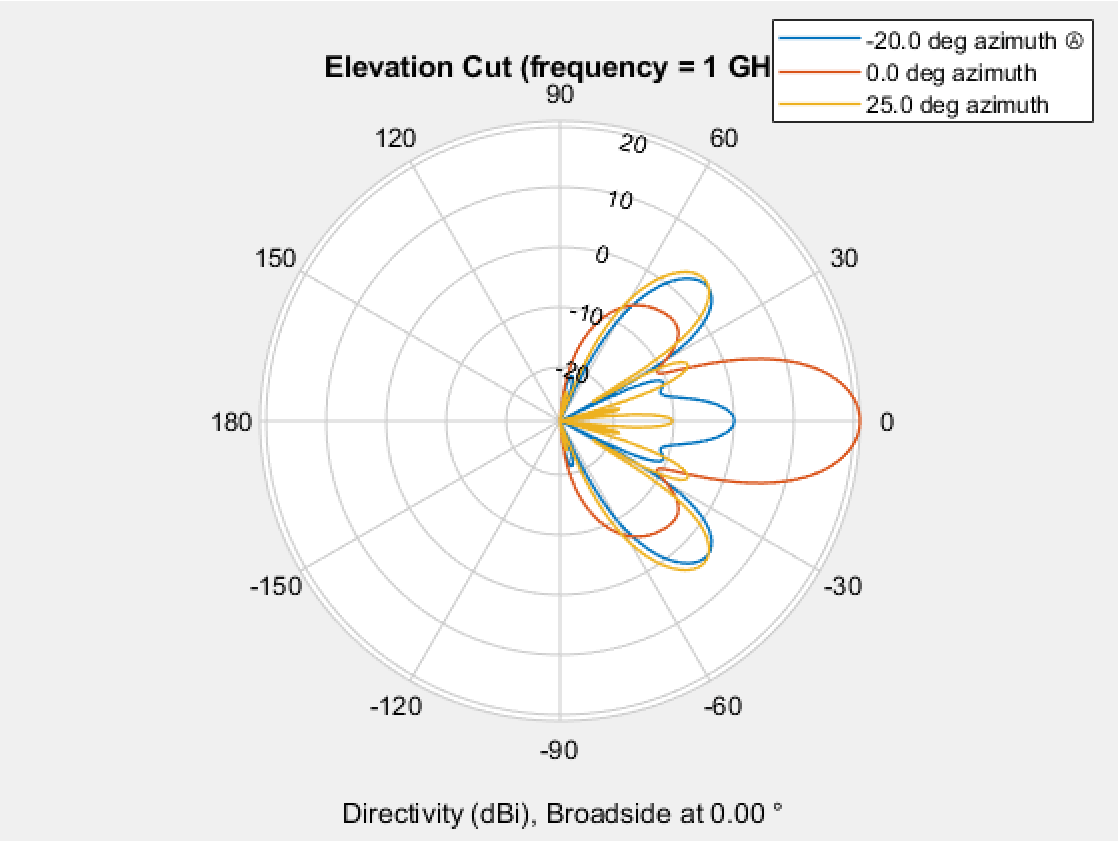


Plot the power pattern

Plot the elevation power pattern of this array for three different azimuth angles: 0, -20 and 25 degrees. Apply radial tapering to the array. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
c = physconst('LightSpeed');
fc = 1e9;
wts = [0.5*ones(N,1); 0.7*ones(N,1); 1*ones(N,1)];
wts = wts/sum(abs(wts));
```

```
patternElevation(sArray,fc,[-20,0,25],'PropagationSpeed',c,...
    'Type','directivity','Weights',wts)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth`

plotResponse

System object: phased.HeterogeneousConformalArray
Package: phased

Plot response pattern of array

Syntax

```
plotResponse(H,FREQ,V)
plotResponse(H,FREQ,V,Name,Value)
hPlot = plotResponse( ___ )
```

Description

plotResponse(H,FREQ,V) plots the array response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in FREQ. The propagation speed is specified in V.

plotResponse(H,FREQ,V,Name,Value) plots the array response with additional options specified by one or more Name,Value pair arguments.

hPlot = plotResponse(___) returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Array object

FREQ

Operating frequency in Hertz specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of H. That property is named FrequencyRange or FrequencyVector, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the 'RespCut' property of H to '3D', FREQ must be a scalar. When FREQ is a row vector, plotResponse draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

CutAngle

Cut angle as a scalar. This argument is applicable only when RespCut is 'Az' or 'El'. If RespCut is 'Az', CutAngle must be between –90 and 90. If RespCut is 'El', CutAngle must be between –180 and 180.

Default: 0

Format

Format of the plot, using one of 'Line', 'Polar', or 'UV'. If you set Format to 'UV', FREQ must be a scalar.

Default: 'Line'

NormalizeResponse

Set this value to true to normalize the response pattern. Set this value to false to plot the response pattern without normalizing it. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: true

OverlayFreq

Set this value to true to overlay pattern cuts in a 2-D line plot. Set this value to false to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is false, FREQ must be a vector with at least two entries.

This parameter applies only when Format is not 'Polar' and RespCut is not '3D'.

Default: true

Polarization

Specify the polarization options for plotting the array response pattern. The allowable values are |'None' | 'Combined' | 'H' | 'V' | where

- 'None' specifies plotting a nonpolarized response pattern
- 'Combined' specifies plotting a combined polarization response pattern
- 'H' specifies plotting the horizontal polarization response pattern
- 'V' specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is 'None'. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: 'None'

RespCut

Cut of the response. Valid values depend on Format, as follows:

- If Format is 'Line' or 'Polar', the valid values of RespCut are 'Az', 'El', and '3D'. The default is 'Az'.

- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

Unit

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

Weights

Weight values applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. The dimension *N* is the number of elements in the array. The interpretation of *M* depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

AzimuthAngles

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

ElevationAngles

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the

Format parameter is set to 'Line' or 'Polar'. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the RespCut parameter to '3D', you can set the ElevationAngles and AzimuthAngles parameters simultaneously.

Default: [-90:90]

UGrid

U coordinate values for plotting array response, specified as a row vector. The UGrid parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to 'U' or '3D'. The values of UGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the UGrid and VGrid parameters simultaneously.

Default: [-1:0.01:1]

VGrid

V coordinate values for plotting array response, specified as a row vector. The VGrid parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to '3D'. The values of VGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set VGrid and UGrid parameters simultaneously.

Default: [-1:0.01:1]

Examples

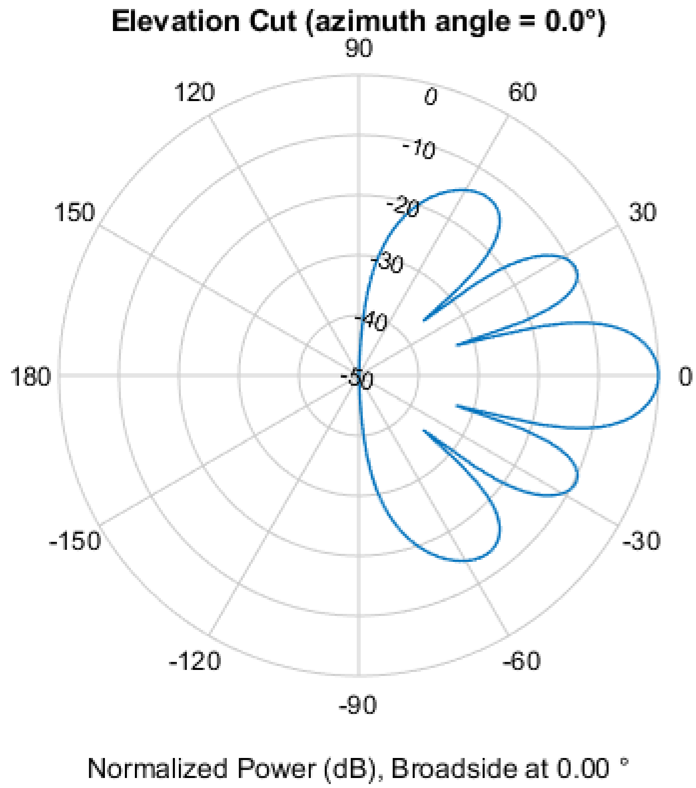
Plot Response and Directivity of 8-Element Uniform Circular Array

This example shows how to construct an 8-element uniform circular array (UCA) with two different antenna patterns.

```
element1 = phased.CosineAntennaElement('CosinePower',1.5);
element2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8;
azang = (0:N-1)*360/N-180;
array = phased.HeterogeneousConformalArray( ...
    'ElementPosition',0.4*[zeros(1,N); cosd(azang); sind(azang)], ...
    'ElementNormal',zeros(2,N),'ElementSet',{element1,element2}, ...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
```

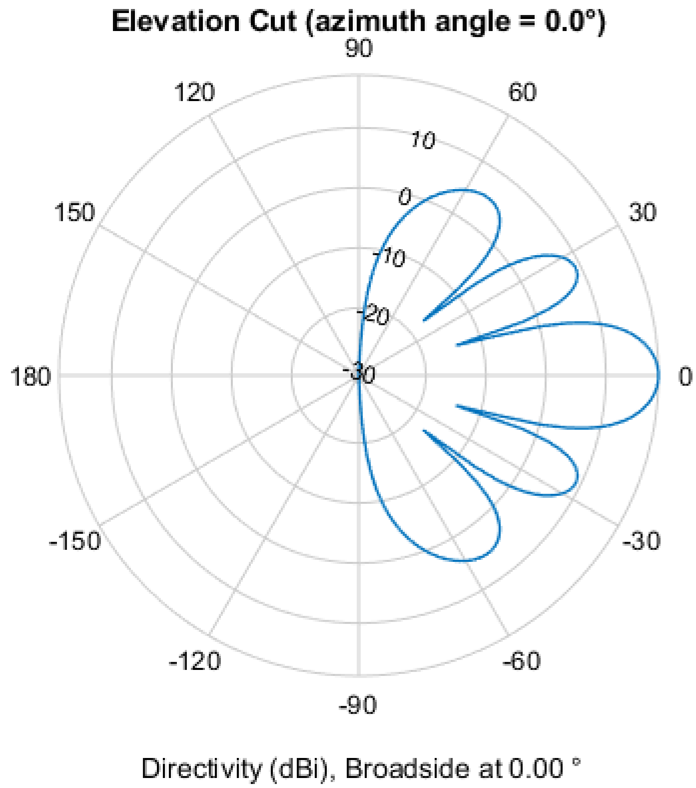
Plot the array elevation response when the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
c = physconst('LightSpeed');
fc = 1e9;
pattern(array,fc,0.0,-90:90,'PropagationSpeed',c,'CoordinateSystem','polar', ...
    'Type','powerdb')
```



Plot the directivity.

```
pattern(array,fc,0.0,-90:90,'PropagationSpeed',c,'CoordinateSystem','polar', ...
    'Type','directivity')
```



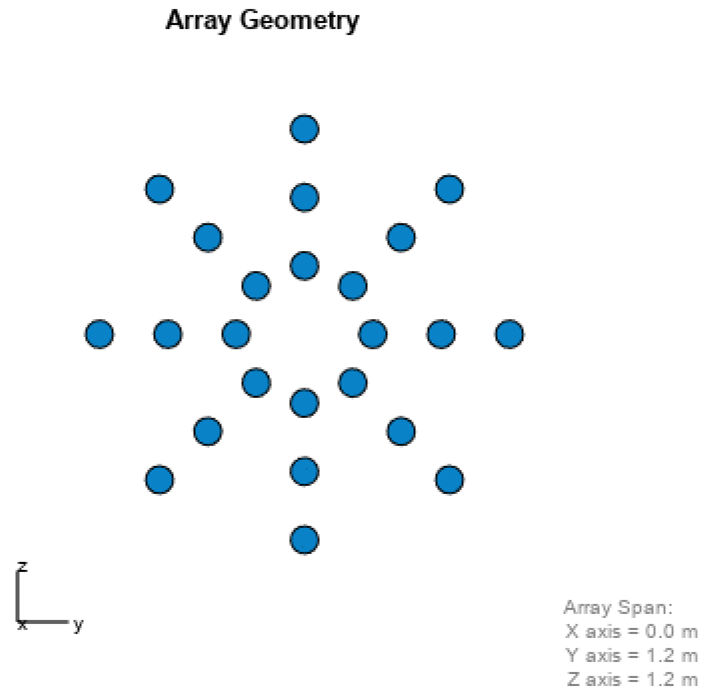
Plot Response of Disk Array

This example shows how to construct a 24-element disk array using elements with two different antenna patterns and plot its response.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8; azang = (0:N-1)*360/N-180;
p0 = [zeros(1,N);cosd(azang);sind(azang)];
posn = [0.6*p0, 0.4*p0, 0.2*p0];
sArray1 = phased.HeterogeneousConformalArray(...
    'ElementPosition',posn,...
    'ElementNormal', zeros(2,3*N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 1 1 1 1,...
    1 1 1 1 1 1 1 1,...
    2 2 2 2 2 2 2 2]);
```

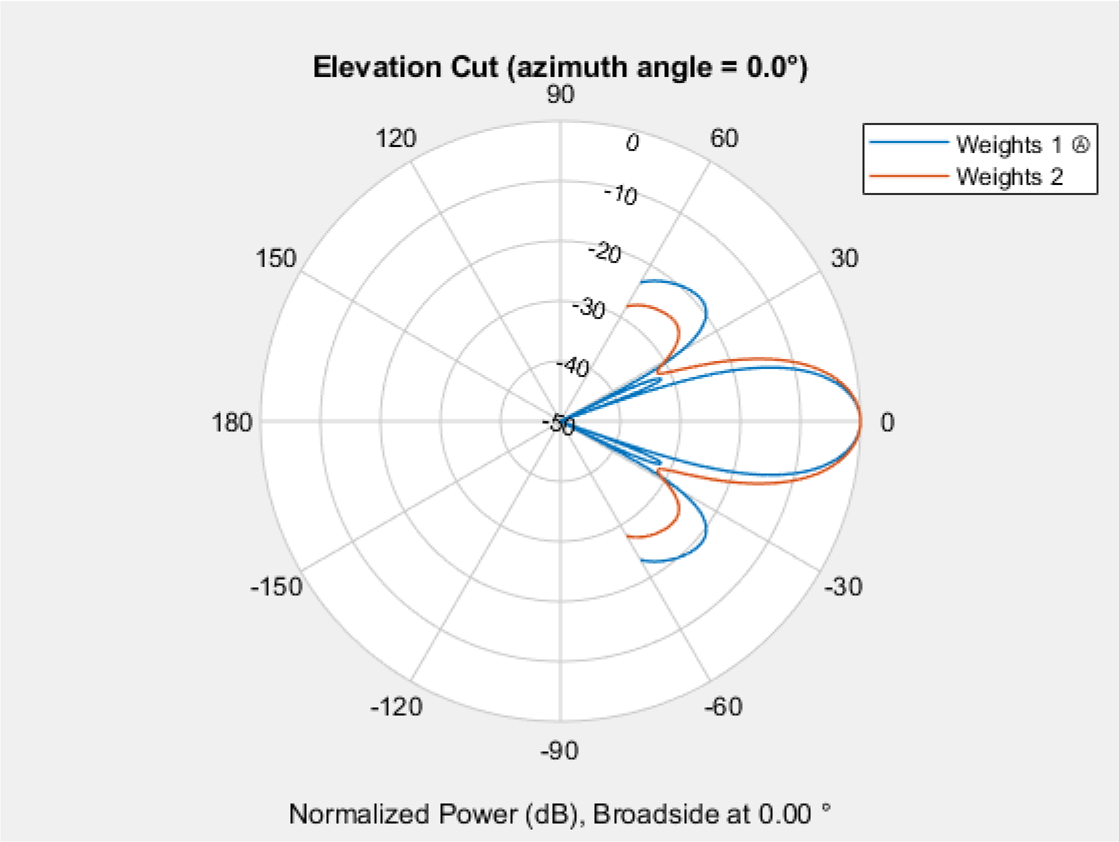
Show the array.

```
viewArray(sArray1);
```



Plot the elevation response of this array using uniform weights on the elements and also a tapered set of weights set by the `Weights` parameter. Using the `ElevationAngles` parameter, restrict the plot of the response from -60 to 60 degrees in 0.1 degree increments. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
c = physconst('LightSpeed');
fc = 1e9;
wts1 = ones(3*N,1);
wts1 = wts1/sum(abs(wts1));
wts2 = [0.5*ones(N,1); 0.7*ones(N,1); 1*ones(N,1)];
wts2 = wts2/sum(abs(wts2));
plotResponse(sArray1,fc,c,'RespCut','El',...
    'Format','Polar',...
    'ElevationAngles',[-60:0.1:60],...
    'Weights',...
    [wts1,wts2],...
    'Unit','db');
```



As expected, the tapered weights broaden the mainlobe and reduce the sidelobes.

See Also

`uv2azel` | `azel2uv`

step

System object: `phased.HeterogeneousConformalArray`
Package: `phased`

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the array elements' responses `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Array object

FREQ

Operating frequencies of array in hertz. `FREQ` is a row vector of length L. Typical values are within the range specified by a property of `H.Element`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has zero response at frequencies outside that range.

ANG

Directions in degrees. `ANG` is either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

If ANG is a row vector of length *M*, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be 0°.

Output Arguments

RESP

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, RESP, has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array. The dimension *M* is the number of angles specified in ANG. *L* is the number of frequencies specified in FREQ. For any element, the columns of RESP contain the responses of the array elements for the corresponding direction specified in ANG. Each of the *L* pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.
- If the array is capable of supporting polarization, the voltage response, RESP, is a MATLAB struct containing two fields, RESP.H and RESP.V. The field, RESP.H, represents the array's horizontal polarization response, while RESP.V represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array, and *M* is the number of angles specified in ANG. *L* is the number of frequencies specified in FREQ. Each column of RESP contains the responses of the array elements for the corresponding direction specified in ANG. Each of the *L* pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.

Examples

Compute Response of Circular Conformal Array

Construct an 8-element uniform circular array using the phased.HeterogeneousConformalArray System object™. Assume the operating frequency is 1 GHz. Find the response of each element in this array in the direction of 30° azimuth and 5°.

```
antenna1 = phased.CosineAntennaElement('CosinePower',1.5);
antenna2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8;
azang = (0:N-1)*360/N-180;
array = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    [cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal', zeros(2,N),...
    'ElementSet',{antenna1,antenna2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
fc = 1e9;
ang = [30;5];
resp = array(fc,ang)
```

```
resp = 8×1

    0.8013
    0.8013
    0.8013
    0.8013
```

```
    0.7666
    0.7666
    0.7666
    0.7666
```

See Also

uv2azel | phitheta2azel

viewArray

System object: `phased.HeterogeneousConformalArray`
Package: `phased`

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

`viewArray(H)` plots the geometry of the array specified in `H`.

`viewArray(H,Name,Value)` plots the geometry of the array, with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = viewArray( ___ )` returns the handle of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

`H`

Array object

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Parent`

Handle to the axes along which the array geometry is displayed.

`ShowNormals`

Set this value to `true` to show the normal directions of all elements of the array. Set this value to `false` to plot the elements without showing normal directions.

Default: `false`

`ShowLocalCoordinates`

Logical flag specifying whether to show the local coordinate axes.

Default: `true`

`ShowAnnotation`

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Default: `true`

`Orientation`

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Default: `[0;0;0]`

`ShowTaper`

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

`ShowIndex`

Vector specifying the element indices to show in the figure. Each number in the vector must be an integer between 1 and the number of elements. You can also specify the value `'All'` to show the indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

`Title`

Character vector specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

`hPlot`

Handle of array elements in figure window.

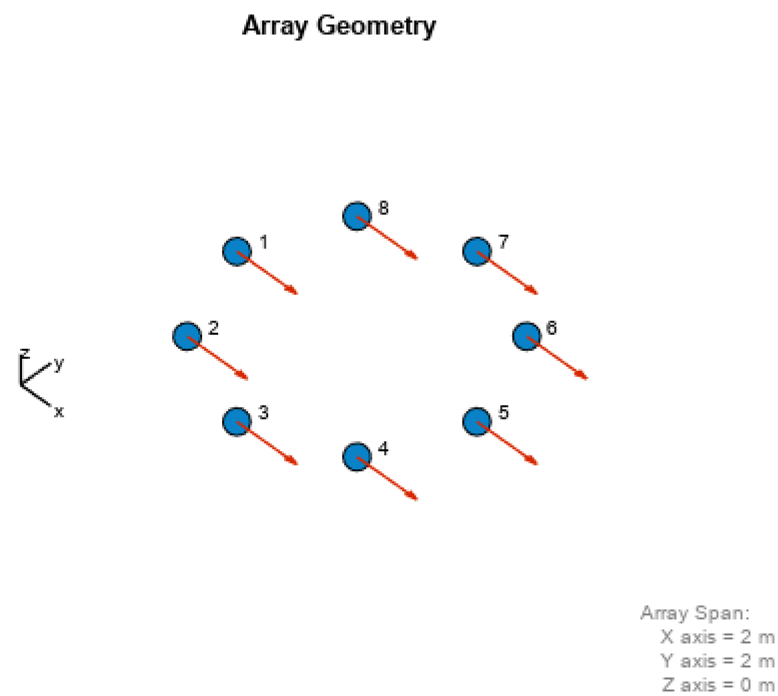
Examples

Element Positions and Normal Directions for Uniform Circular Array

Display the element positions and normal directions for all elements of an 8-element heterogeneous uniform circular array.

Create the elements and the array.

```
antenna1 = phased.CosineAntennaElement('CosinePower',1.5);
antenna2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8;
azang = (0:N-1)*360/N-180;
array = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    [cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal', zeros(2,N),...
    'ElementSet',{antenna1,antenna2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
viewArray(array,'ShowIndex','all','ShowNormal',true);
```



See Also

phased.ArrayResponse

Topics

Phased Array Gallery

phased.HeterogeneousULA

Package: `phased`

Heterogeneous uniform linear array

Description

The `phased.HeterogeneousULA` object creates a uniform linear array from a heterogeneous set of antenna elements. A heterogeneous array is an array in which the antenna or microphone elements may be of different kinds or have different properties. An example would be an array of elements each having different antenna patterns.

To compute the response for each element in the array for specified directions:

1. Define and set up your uniform linear array. See “Construction” on page 1-551.
2. Call `step` to compute the response according to the properties of `phased.HeterogeneousULA`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.HeterogeneousULA` creates a heterogeneous uniform linear array (ULA) System object, `H`. The object models a heterogeneous ULA formed with generally different sensor elements. The origin of the local coordinate system is the phase center of the array. The positive *x*-axis is the direction normal to the array, and the elements of the array are located along the *y*-axis.

`H = phased.HeterogeneousULA(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`ElementSet`

Set of elements used in the array

Set of elements used in the sensor array, specified as a row MATLAB cell array. Elements specified in the `ElementSet` property must be either Phased Array System Toolbox antennas, microphones, or transducers System objects or Antenna Toolbox System object. In addition, all specified antenna elements must have the same polarization capability.

Default: One cell containing one isotropic antenna element

`ElementIndices`

Elements location assignment

This property specifies the mapping of elements in the array. The property assigns elements to their locations in the array using indices into the `ElementSet` property. `ElementIndices` must be a 1-by-*N* row vector where *N* is greater than 1. *N* is the number of elements in the sensor array. The values in `ElementIndices` should be less than or equal to the number of entries in the `ElementSet` property.

Default: `[1 1]`

`ElementSpacing`

Element spacing

A scalar containing the spacing (in meters) between two adjacent elements in the array.

Default: `0.5`

`ArrayAxis`

Array axis

Array axis, specified as one of `'x'`, `'y'`, or `'z'`. ULA array elements are located along the selected coordinate system axis.

Element normal vectors are determined by the selected array axis

ArrayAxis Property Value	Element Normal Direction
`'x'`	azimuth = 90°, elevation = 0° (*y*-axis)
`'y'`	azimuth = 0°, elevation = 0° (*x*-axis)
`'z'`	azimuth = 0°, elevation = 0° (*x*-axis)

Default: `'y'`

`Taper`

Element tapering

Element tapering or weighting, specified as a complex-valued scalar, 1-by-*N* row vector, or *N*-by-1 column vector. The quantity *N* is the number of elements in the array as determined by the size of the `ElementIndices` property. Tapers, also known as weights, are applied to each sensor element in the sensor array and modify both the amplitude and phase of the received data. If `'Taper'` is a scalar, the same taper value is applied to all elements. If `'Taper'` is a vector, each taper value is applied to the corresponding sensor element.

Default: `1`

Methods

Specific to `phased.HeterogeneousULA` Object	
`beamwidth`	Compute and display beamwidth of an array
`collectPlaneWave`	Simulate received plane waves

Specific to phased.HeterogeneousULA Object	
directivity	Directivity of heterogeneous uniform linear array
getElementNormal	Normal vector for array elements
getElementPosition	Positions of array elements
getNumElements	Number of elements in array
getTaper	Array element tapers
isPolarizationCapable	Polarization capability
pattern	Plot heterogeneous ULA pattern
patternAzimuth	Plot heterogeneous ULA directivity or pattern versus azimuth
patternElevation	Plot heterogeneous ULA array directivity or pattern versus elevation
perturbations	Perturbations defined on phased array
perturbedArray	Apply perturbations to phased array
perturbedPattern	Compute and plot azimuth pattern of perturbed array
step	Output responses of array elements
viewArray	View array geometry

Common to All System Objects	
release	Allow System object property value changes

Examples

Power Pattern of 10-Element Heterogeneous ULA Array

Create a 10-element heterogeneous ULA consisting of cosine antenna elements with different power exponents. Two elements at each end have power values of 1.5 while the inside elements have power exponents of 1.8. Find the power pattern in dB of each element at boresight.

Construct the heterogeneous array and show the element responses at 1 GHz.

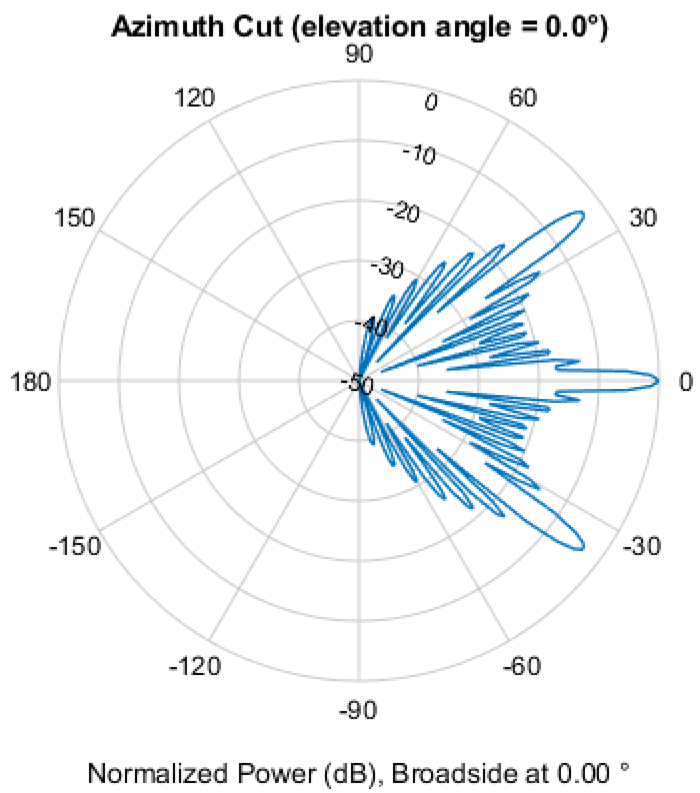
```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 2 2 2 2 2 2 1 1 ]);
fc = 1e9;
```

```
ang = [0;0];
resp = step(sArray,fc,ang)
```

```
resp =

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
```

Plot an azimuth cut of the array response at 1 GHz.

```
c = physconst('LightSpeed');
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar');
pattern(sArray,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','powerdb');
```



Pattern of Array of Polarized Short-Dipole Antennas

Construct a heterogeneous uniform line array of 10 short-dipole sensor elements. Because short dipoles support polarization, the array should also. Verify that the array supports polarization by looking at the output of `isPolarizationCapable`. Then, draw the array, showing the tapering.

Construct the array

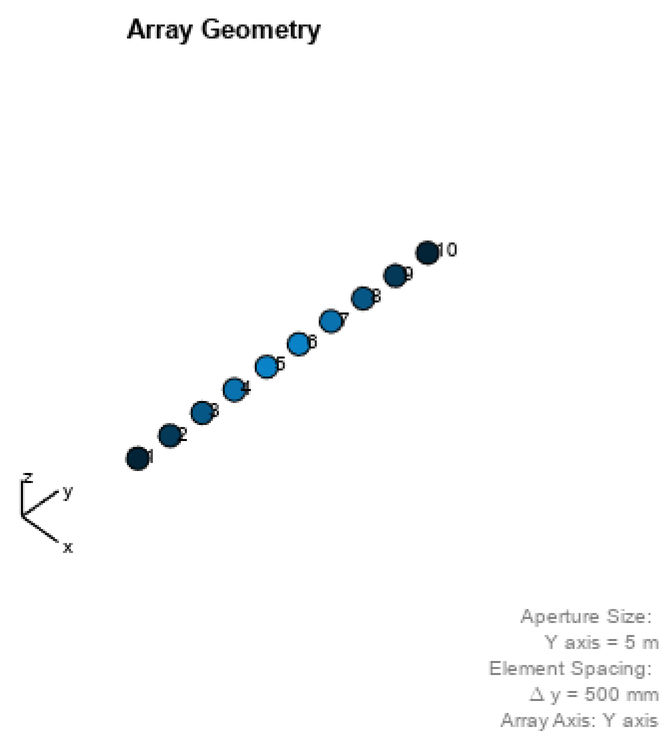
Construct the array. Then, verify that it supports polarization by looking at the returned value of the `isPolarizationCapable` method.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 2 2 2 2 2 2 1 1 ],...
    'Taper',taylorwin(10)');
isPolarizationCapable(sArray)
```

```
ans = logical
   1
```

View the array

```
viewArray(sArray,'ShowTaper',true,'ShowIndex',...
    'All','ShowTaper',true)
```



Show the response

Show the element horizontal polarization responses at 10 degrees azimuth angle.

```
fc = 150e6;
ang = [10];
resp = step(sArray,fc,ang)
```

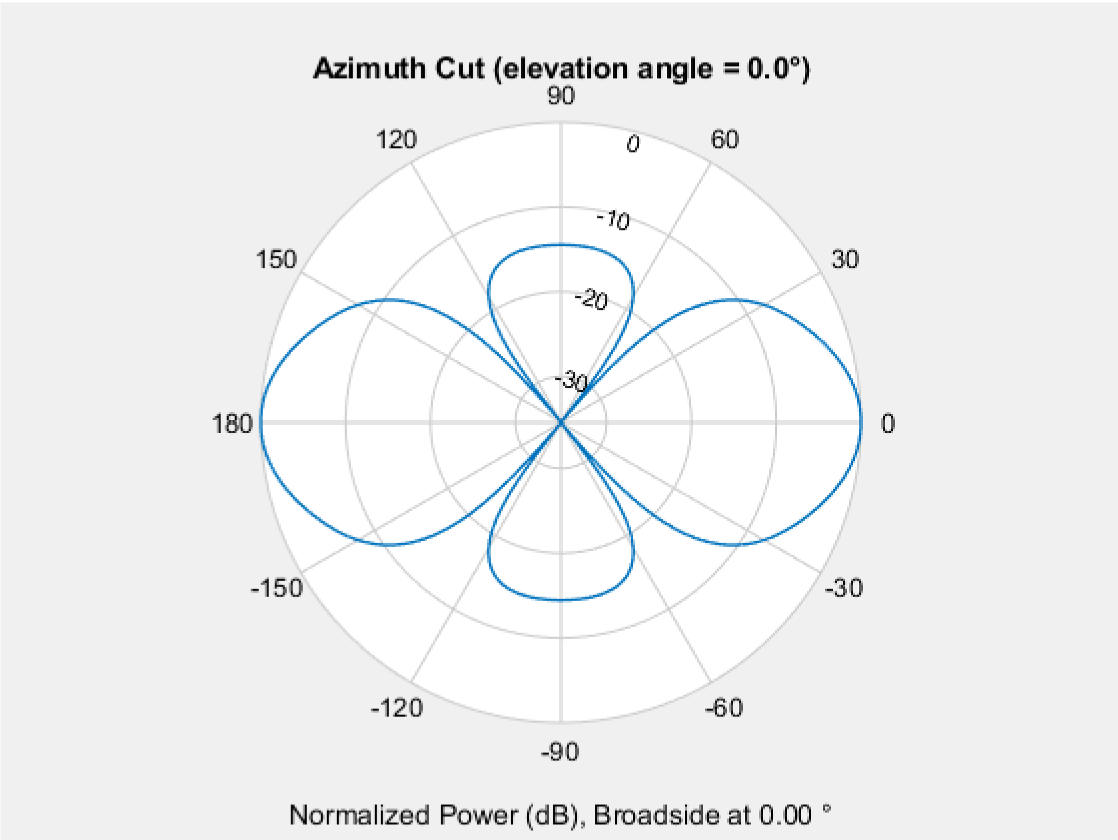
```
resp = struct with fields:
    H: [10x1 double]
    V: [10x1 double]
```

```
resp.H
```

```
ans = 10×1

         0
         0
   -1.2442
   -1.6279
   -1.8498
   -1.8498
   -1.6279
   -1.2442
         0
         0
```

Plot the combined polarization response

```
c = physconst('LightSpeed');
pattern(sArray,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Polarization','combined');
```



Version History

Introduced in R2013a

References

[1] Brookner, E., ed. *Radar Technology*. Lexington, MA: LexBook, 1996.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, `plotResponse`, and `viewArray` methods are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.ULA` | `phased.UCA` | `phased.URA` | `phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.HeterogeneousURA` | `phased.HeterogeneousURA` | `phased.CosineAntennaElement` | `phased.CrossedDipoleAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ShortDipoleAntennaElement`

Topics

“Phased Array Gallery”

directivity

System object: `phased.HeterogeneousULA`
Package: `phased`

Directivity of heterogeneous uniform linear array

Syntax

```
D = directivity(H,FREQ,ANGLE)
D = directivity(H,FREQ,ANGLE,Name,Value)
```

Description

`D = directivity(H,FREQ,ANGLE)` computes the “Directivity (dBi)” on page 1-562 of a heterogeneous uniform linear array of antenna or microphone elements, `H`, at frequencies specified by the `FREQ` and in angles of direction specified by the `ANGLE`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`D = directivity(H,FREQ,ANGLE,Name,Value)` computes the directivity with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`H` — Heterogeneous uniform linear array

System object

Heterogeneous uniform linear array, specified as a `phased.HeterogeneousULA` System object.

Example: `H = phased.HeterogeneousULA;`

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`ANGLE` — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Heterogeneous Uniform Linear Array

Compute the directivity of a 10-element heterogeneous ULA consisting of cosine antenna elements with different power factors. The two elements at each end have power values of 1.5 while the inner elements have power values of 1.8.

Construct the heterogeneous array. Set the signal frequency to 1 GHz.

```
c = physconst('LightSpeed');
freq = 1e9;
ang = [30;0];
lambda = c/freq;
```

Create the cosine antenna elements.

```
myElement1 = phased.CosineAntennaElement;
myElement1.CosinePower = 1.5;
myElement2 = phased.CosineAntennaElement;
myElement2.CosinePower = 1.8;
```

Create the Heterogeneous ULA.

```
myArray = phased.HeterogeneousULA;
myArray.ElementSet = {myElement1,myElement2};
myArray.ElementIndices = [1 1 2 2 2 2 2 2 1 1 ];
myArray.ElementSpacing = 0.5*lambda;
```

Create the steering vector and compute the directivity in the same direction as the steering vector.

```
w = steervec(getElementPosition(myArray)/lambda,ang);
d = directivity(myArray,freq,ang,'PropagationSpeed',c,...
    'Weights',w)
```

```
d = 17.0102
```

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternAzimuth` | `patternElevation`

collectPlaneWave

System object: phased.HeterogeneousULA
Package: phased

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

Y = collectPlaneWave(H,X,ANG) returns the received signals at the sensor array, H, when the input signals indicated by X arrive at the array from the directions specified in ANG.

Y = collectPlaneWave(H,X,ANG,FREQ), in addition, specifies the incoming signal carrier frequency in FREQ.

Y = collectPlaneWave(H,X,ANG,FREQ,C), in addition, specifies the signal propagation speed in C.

Input Arguments

H

Array object.

X

Incoming signals, specified as an M-column matrix. Each column of X represents an individual incoming signal.

ANG

Directions from which incoming signals arrive, in degrees. ANG can be either a 2-by-M matrix or a row vector of length M.

If ANG is a 2-by-M matrix, each column specifies the direction of arrival of the corresponding signal in X. Each column of ANG is in the form [azimuth; elevation]. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If ANG is a row vector of length M, each entry in ANG specifies the azimuth angle. In this case, the corresponding elevation angle is assumed to be 0°.

FREQ

Carrier frequency of signal in hertz. FREQ must be a scalar.

Default: 3e8

c

Propagation speed of signal in meters per second.

Default: Speed of light

Output Arguments

Y

Received signals. Y is an N-column matrix, where N is the number of elements in the array H. Each column of Y is the received signal at the corresponding array element, with all incoming signals combined.

Examples

Simulate Received Signals at Heterogeneous ULA

Simulate two received signal at a heterogeneous 4-element ULA. The signals arrive from 10° and 30° degrees azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
array = phased.HeterogeneousULA('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 2 2 1]);
```

Create a random plane wave signal.

```
y = collectPlaneWave(array,randn(4,2),[10 30],1e8,physconst('LightSpeed'));
```

Display the signal at the first element.

```
y(:,1)
```

```
ans = 4×1 complex

   0.7430 - 0.3705i
   0.8418 + 0.4308i
  -2.4817 + 0.9157i
   1.0724 - 0.4748i
```

Algorithms

collectPlaneWave modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. The method does not account for the response of individual elements in the array.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

getElementNormal

System object: `phased.HeterogeneousULA`
Package: `phased`

Normal vector to array elements

Syntax

```
normvec = getElementNormal(sULA)
normvec = getElementNormal(sULA,elemidx)
```

Description

`normvec = getElementNormal(sULA)` returns the normal vectors of the array elements of the `phased.HeterogeneousULA` System object, `sULA`. The output argument `normvec` is a 2-by-*N* matrix, where *N* is the number of elements in array, `sULA`. Each column of `normvec` defines the normal direction of an element in the local coordinate system in the form`[az;el]`. Units are degrees. The origin of the local coordinate system is defined by the phase center of the array.

`normvec = getElementNormal(sULA,elemidx)` returns only the normal vectors of the elements specified in the element index vector, `elemidx`. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

`sULA` — Uniform line array

`phased.HeterogeneousULA` System object

Uniform line array, specified as a `phased.HeterogeneousULA` System object.

Example: `sULA = phased.HeterogeneousULA`

`elemidx` — Element indices

all array elements (default) | integer-valued 1-by-*M* row vector | integer-valued *M*-by-1 column vector

Element indices , specified as a 1-by-*M* or *M*-by-1 vector. Index values lie in the range 1 to *N* where *N* is the number of elements of the array. When `elemidx` is specified, `getElementNormal` returns the normal vectors of the elements contained in `elemidx`.

Example: `[1,5,4]`

Output Arguments

`normvec` — Element normal vectors

2-by-*P* real-valued vector

Element normal vectors, specified as a 2-by-*P* real-valued vector. Each column of `normvec` takes the form `[az,el]`. When `elemidx` is not specified, *P* equals the array dimension. When `elemidx` is specified, *P* equals the length of `elemidx`, *M*.

Examples

Heterogeneous ULA Element Normals

Construct three 5-element heterogeneous ULA's with elements along the *x*-, *y*-, and *z*-axes. Obtain the element normals.

Create two types of cosine antennas.

```
sCosAnt1 = phased.CosineAntennaElement('CosinePower',[1.5,1.5]);
sCosAnt2 = phased.CosineAntennaElement('CosinePower',[1.8,1.8]);
```

First, choose the array axis to lie along the *x*-axis.

```
sULA1 = phased.HeterogeneousULA('ElementSet',{sCosAnt1,sCosAnt2},...
    'ElementIndices',[1 2 2 2 1],'ArrayAxis','x');
norm = getElementNormal(sULA1)
```

```
norm = 2×5

    90    90    90    90    90
     0     0     0     0     0
```

The element normal vectors point along the *y*-axis.

Next, choose the array axis along the *y*-axis.

```
sULA2 = phased.HeterogeneousULA('ElementSet',{sCosAnt1,sCosAnt2},...
    'ElementIndices',[1 2 2 2 1],'ArrayAxis','y');
norm = getElementNormal(sULA2)
```

```
norm = 2×5

     0     0     0     0     0
     0     0     0     0     0
```

The element normal vectors point along the *x*-axis.

Finally, set the array axis along the *z*-axis. Obtain the normal vectors of the odd-numbered elements.

```
sULA3 = phased.HeterogeneousULA('ElementSet',{sCosAnt1,sCosAnt2},...
    'ElementIndices',[1 2 2 2 1],'ArrayAxis','z');
norm = getElementNormal(sULA3,[1,3,5])
```

```
norm = 2×3

     0     0     0
     0     0     0
```

The element normal vectors also point along the *x*-axis.

Version History

Introduced in R2016a

getElementPosition

System object: phased.HeterogeneousULA
Package: phased

Positions of array elements

Syntax

```
pos = getElementPosition(sHULA)
pos = getElementPosition(sHULA,elemidx)
```

Description

`pos = getElementPosition(sHULA)` returns the element positions of the `phased.HeterogeneousULA` System object, `sHULA`. `pos` is a 3-by-N matrix, where N is the number of elements in `sHULA`. Each column of `pos` defines the position of an element in the local coordinate system, in meters, using the form `[x;y;z]`. The origin of the local coordinate system is the phase center of the array. The positions of the array elements depend upon the value of the `ArrayAxis` property.

`pos = getElementPosition(sHULA,elemidx)` returns only the positions of the elements that are specified in the element index vector `elemidx`. This syntax can use any of the input arguments in the previous syntax.

Examples

Position of Heterogeneous ULA Elements

Construct a 4-element heterogeneous ULA of different types of short-dipole antenna elements. Then, obtain the element positions.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 1]);
pos = getElementPosition(sArray)
```

```
pos = 3×4

         0         0         0         0
   -0.7500   -0.2500    0.2500    0.7500
         0         0         0         0
```

getNumElements

System object: `phased.HeterogeneousULA`
Package: `phased`

Number of elements in array

Syntax

`N = getNumElements(array)`

Description

`N = getNumElements(array)` returns the number of elements, *N*, in the heterogeneous ULA object `array`.

Examples

Number of Elements of Heterogeneous ULA

Construct a 4-element heterogeneous ULA. Then verify the number of elements in the array.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
array = phased.HeterogeneousULA('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 2 2 1]);
N = getNumElements(array)
```

```
N = 4
```

getTaper

System object: phased.HeterogeneousULA
Package: phased

Array element tapers

Syntax

`wts = getTaper(array)`

Description

`wts = getTaper(array)` returns the tapers, `wts`, applied to each element of the phased heterogeneous uniform line array (ULA), `h`. Tapers are often referred to as weights.

Input Arguments

`array` — Heterogeneous uniform line array
`phased.HeterogeneousULA` System object

Heterogeneous uniform line array, specified as a `phased.HeterogeneousULA` System object.

Output Arguments

`wts` — Array element tapers
N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1 complex-valued vector, where *N* is the number of elements in the array.

Examples

Heterogeneous ULA with Taylor Window Taper

Construct a 5-element heterogeneous ULA with a Taylor window taper. The array consists of short-dipole antenna elements with different orientations.Then, obtain the element taper values.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
array = phased.HeterogeneousULA('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 2 2 2 1],'Taper',taylorwin(5)');
w = getTaper(array)
```

```
w = 5×1

    0.5181
    1.2029
```

```
1.5581
1.2029
0.5181
```

isPolarizationCapable

System object: `phased.HeterogeneousULA`
Package: `phased`

Polarization capability

Syntax

`flag = isPolarizationCapable(array)`

Description

`flag = isPolarizationCapable(array)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

`array` — Heterogeneous uniform line array
`phased.HeterogeneousULA` System object

Heterogeneous uniform line array, specified as a `phased.HeterogeneousULA` System object.

Output Arguments

`flag` — Polarization-capability flag

Polarization-capability flag, returned as a Boolean value `1` if the array supports polarization or `0` if it does not.

Examples

Heterogeneous ULA of Short-Dipole Antenna Elements Supports Polarization

Show that a heterogeneous array of short-dipole antenna elements supports polarization.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
array = phased.HeterogeneousULA('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 2 2 2 1]);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

pattern

System object: `phased.HeterogeneousULA`
Package: `phased`

Plot heterogeneous ULA pattern

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sArray,FREQ)` plots the 3-D array directivity pattern (in dBi) for the array specified in `sArray`. The operating frequency is specified in `FREQ`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`pattern(sArray,FREQ,AZ)` plots the array directivity pattern at the specified azimuth angle.

`pattern(sArray,FREQ,AZ,EL)` plots the array directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the array pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See “Convert plotResponse to pattern” on page 1-581 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sArray` — Heterogeneous ULA
System object

Heterogeneous conformal array, specified as a `phased.HeterogeneousULA` System object.

Example: `sArray= phased.HeterogeneousULA;`

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Orientation` — Array orientation
`[0;0;0]`. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

`Normalize` — Display normalize pattern
`true` (default) | `false`

Display normalized pattern, specified as the comma-separated pair consisting of `'Normalize`' and a Boolean. Set this parameter to `true` to display a normalized pattern. This parameter does not apply when you set `'Type'` to `'directivity'`. Directivity patterns are already normalized.

Data Types: `logical`

`ShowArray` — View array geometry
`false` (default) | `true`

View the array geometry along with the 3D radiation pattern, specified as `false` or `true`.

Data Types: `logical`

`ShowLocalCoordinates` — Show local coordinate axes
`true` (default) | `false`

Show the local coordinate axes, specified as `true` or `false`.

Data Types: `logical`

`ShowColorbar` — Show colorbar
`true` (default) | `false`

Show the colorbar, specified as `true` or `false`.

Data Types: `logical`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

PAT — Array pattern
M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of `PAT` correspond to the dimensions of the output arguments `AZ_ANG` and `EL_ANG`.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in `AZ`. The columns of `PAT` correspond to the values in `AZ_ANG`. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

Examples

Azimuth Power Pattern For Two Frequencies

Create a 5-element heterogeneous ULA from short-dipole antenna elements with different axis directions. Draw the azimuth power pattern for the horizontal polarization component at 0 degrees elevation for two frequencies, 300 MHz and 400 MHz.

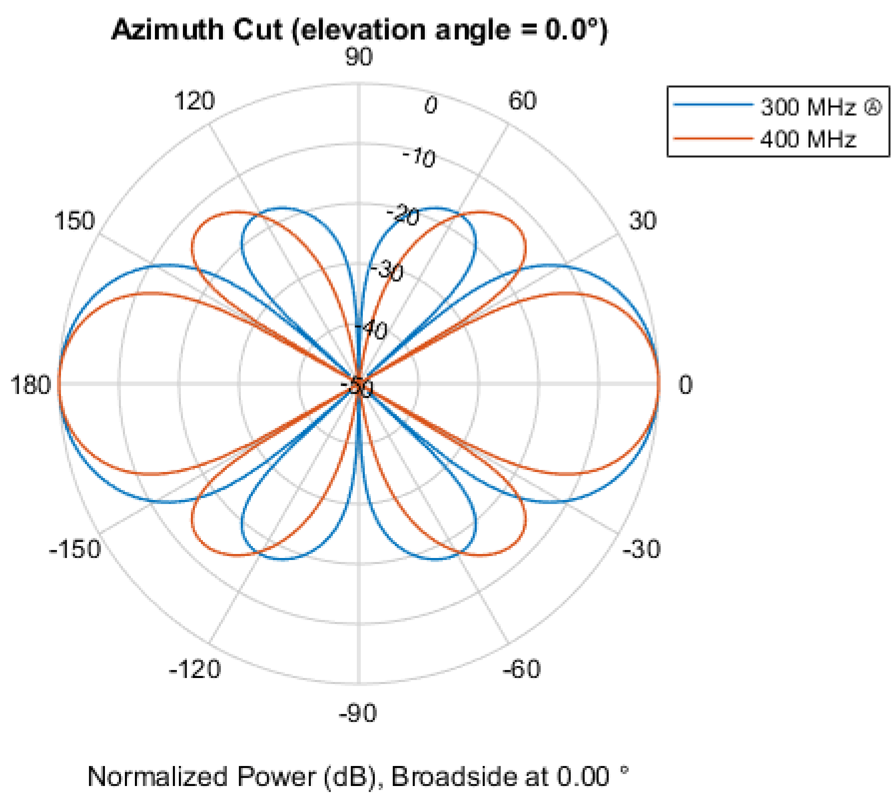
Construct Heterogeneous ULA

Construct the array from z-directed and y-directed short dipole antenna elements.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
```

Plot the patterns

```
fc = [300e6 400e6];
c = physconst('LightSpeed');
pattern(sArray,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'PlotStyle','overlay',...
    'Polarization','H')
```



Directivity Pattern in UV Space

Create an 11-element heterogeneous ULA from short-dipole antenna elements with different axis directions. Draw the 3-D power pattern for the horizontal polarization component at 300 MHz.

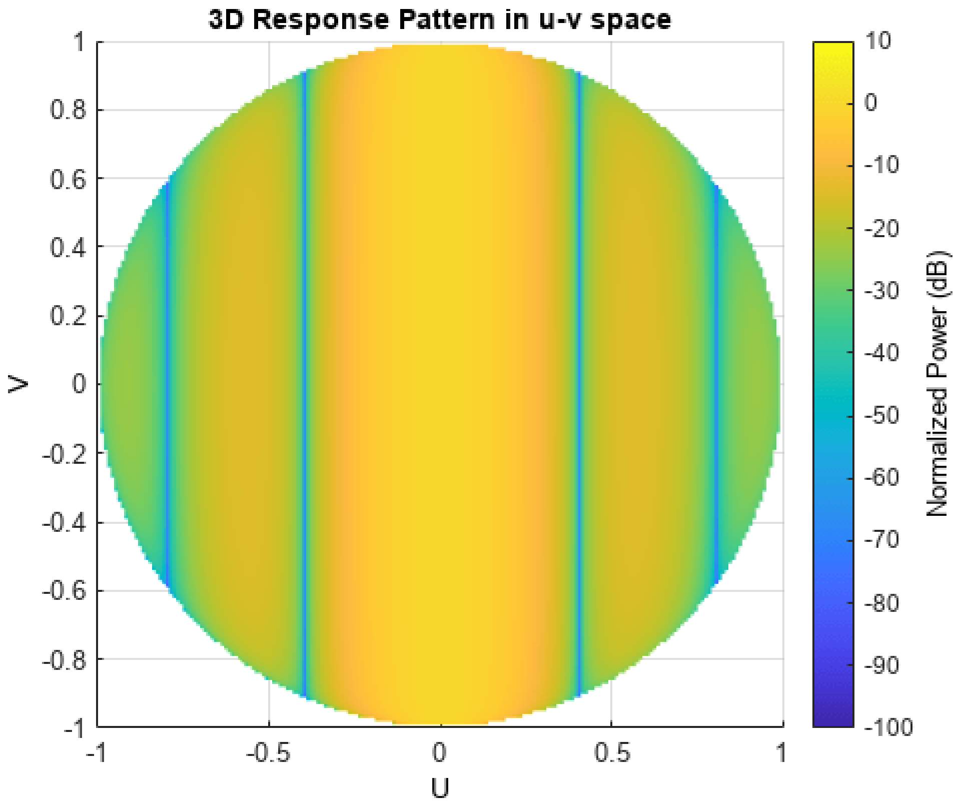
Construct Heterogeneous ULA

Construct the array from z-directed and y-directed short dipole antenna elements.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 2 2 2 2 2 1 1 1]);
```

Plot the patterns

```
fc = 300e6;
c = physconst('LightSpeed');
pattern(sArray,fc,-1:.01:1,-1:.01:1,...
    'PropagationSpeed',c,...
    'CoordinateSystem','uv',...
    'Type','powerdb',...
    'Polarization','H')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td><code>'Format'</code> and <code>'RespCut'</code> name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using <code>plotResponse</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'RespCut'</code> to <code>'Az'</code> or <code>'El'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using either the <code>'AzimuthAngles'</code> or <code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'RespCut'</code> to <code>'3D'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using both the <code>'AzimuthAngles'</code> and<code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'RespCut'</code> to<code>'U'</code>. Set <code>'Format'</code> to <code>'UV'</code>. Set the display range using the <code>'UGrid'</code> name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'RespCut'</code> to<code>'3D'</code>. Set <code>'Format'</code> to</td></tr>
</table>
</td>
<td><code>'CoordinateSystem'</code> name-value pair used together with the <code>AZ</code> and <code>EL</code> input arguments.

<code>'CoordinateSystem'</code> has the same options as the <code>plotResponse</code> method <code>'Format'</code>name-value pair, except that <code>'line'</code> is now named <code>'rectangular'</code>. The table shows how to create different types of plots using <code>pattern</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify either <code>AZ</code> or <code>EL</code> as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify both <code>AZ</code> and <code>EL</code> as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space vector.</td></tr>
</table>
If you set <code>CoordinateSystem</code> to <code>'uv'</code>, enter the UV grid values using <code>AZ</code> and <code>EL</code>.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `patternElevation`

patternAzimuth

System object: `phased.HeterogeneousULA`
Package: `phased`

Plot heterogeneous ULA directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sArray,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(sArray,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sArray,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sArray` — Heterogeneous ULA

System object

Heterogeneous ULA, specified as a `phased.HeterogeneousULA` System object.

Example: `sArray= phased.HeterogeneousULA;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

Azimuth — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

Examples

Azimuth Directivity Pattern For Steered Array

Create an 11-element heterogeneous ULA from short-dipole antenna elements with different axis directions. The element spacing is 0.4 meters. Draw the azimuthal directivity pattern for 0 degrees elevation at an operating frequency of 300 MHz. Then, steer the array and draw the azimuthal directivity pattern.

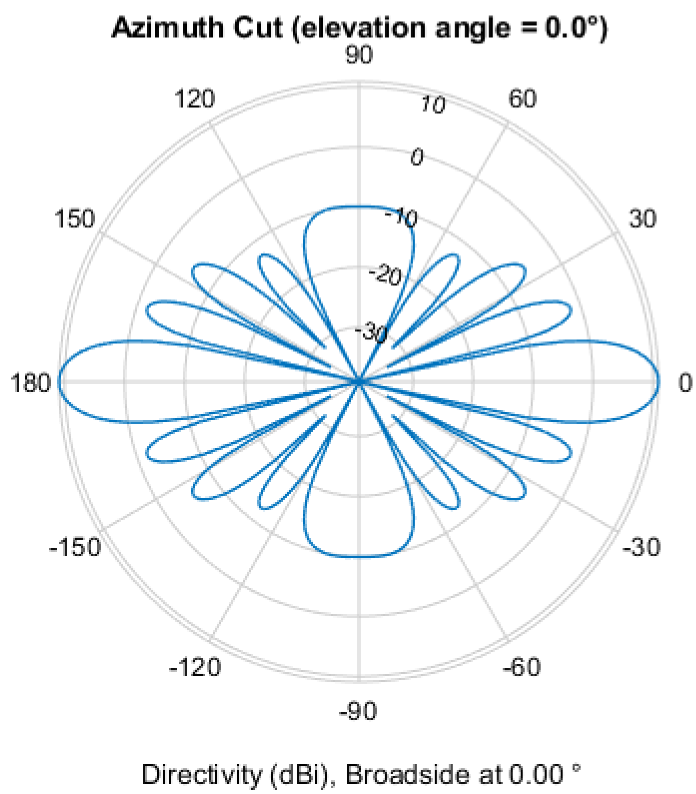
Construct Heterogeneous ULA

Construct the array from z-directed and y-directed short dipole antenna elements.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[200e6 500e6],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[200e6 500e6],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSpacing',0.4,...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 2 2 2 2 2 1 1 1]);
```

Plot Directivity Pattern

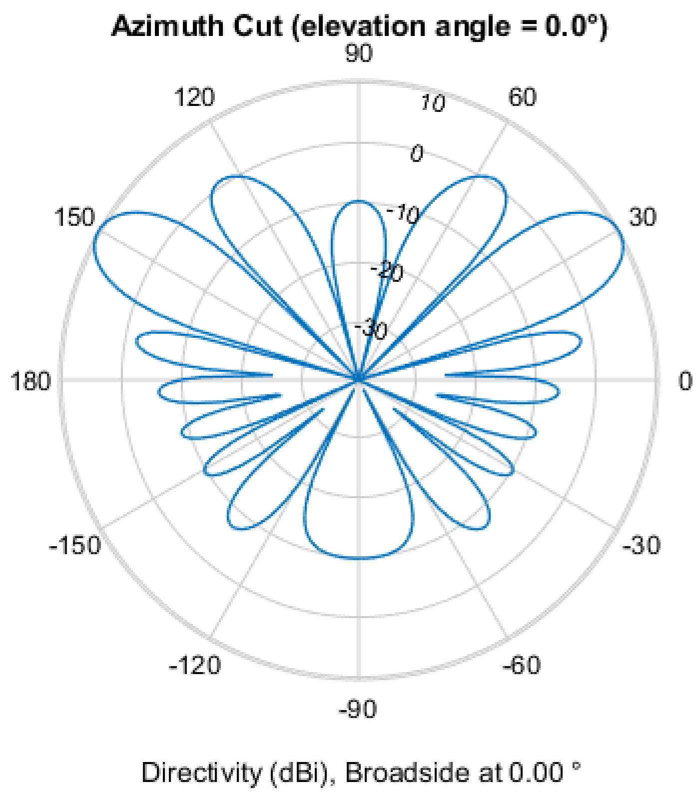
```
fc = 300e6;
c = physconst('LightSpeed');
lam = c/fc;
patternAzimuth(sArray,fc,0,...
    'PropagationSpeed',c,...
    'Type','directivity')
```



Steer Array and Plot Directivity Pattern

Steer the array to 30 degrees in azimuth by applying weights to achieve a linear phase shift.

```
theta = 30;
d = [0:10]*0.4;
ph = 2*pi*d'/lam*sind(theta);
wts = exp(1i*ph);
patternAzimuth(sArray,fc,0,...
    'PropagationSpeed',c,...
    'Type','directivity',....
    'Weights',wts)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta,\varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: phased.HeterogeneousULA
Package: phased

Plot heterogeneous ULA directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sArray,FREQ)` plots the 2-D array directivity pattern versus elevation (in dBi) for the array `sArray` at zero degrees azimuth angle. When `AZ` is a vector, multiple overlaid plots are created. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternElevation(sArray,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sArray,FREQ,AZ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sArray` — Heterogeneous ULA

System object

Heterogeneous ULA array, specified as a `phased.HeterogeneousULA` System object.

Example: `sArray= phased.HeterogeneousULA;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

`PAT` — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Elevation Power Pattern for Two Azimuth Directions

Create an 11-element heterogeneous ULA from short-dipole antenna elements with different axis directions. The element spacing is 0.4 meters. Draw the elevation power pattern for 0 and 30 degrees azimuth for 300 MHz.

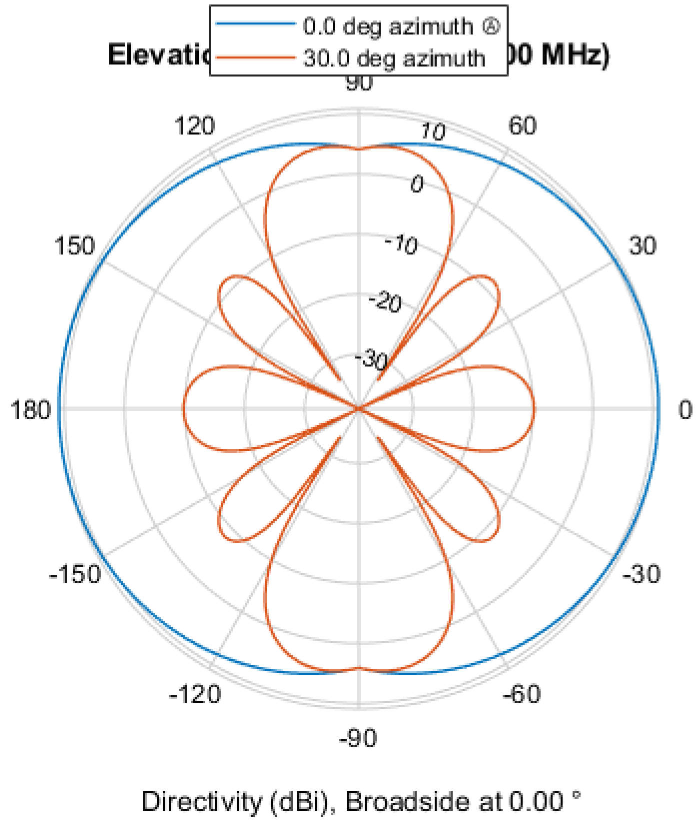
Construct Heterogeneous ULA

Construct the array from z-directed and y-directed short dipole antenna elements.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[200e6 500e6],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[200e6 500e6],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSpacing',0.4,...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 2 2 2 2 2 1 1 1]);
```

Plot Directivity Pattern

```
fc = 300e6;
c = physconst('LightSpeed');
patternElevation(sArray,fc,[0,30],...
    'PropagationSpeed',c,...
    'Type','directivity')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth`

plotResponse

System object: `phased.HeterogeneousULA`
Package: `phased`

Plot response pattern of array

Syntax

```
plotResponse(H,FREQ,V)
plotResponse(H,FREQ,V,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ,V)` plots the array response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`. The propagation speed is specified in `V`.

`plotResponse(H,FREQ,V,Name,Value)` plots the array response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

`H`

Array object

`FREQ`

Operating frequency in Hertz specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

`V`

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CutAngle`

Cut angle as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

`Format`

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

`NormalizeResponse`

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

`OverlayFreq`

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

`Polarization`

Specify the polarization options for plotting the array response pattern. The allowable values are `|'None' | 'Combined' | 'H' | 'V' |` where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

`RespCut`

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.

- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

Unit

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
db	power pattern in dB scale
mag	field pattern
pow	power pattern
dbi	directivity

Default: `'db'`

Weights

Weight values applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. The dimension *N* is the number of elements in the array. The interpretation of *M* depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

AzimuthAngles

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

ElevationAngles

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the

`Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

`UGrid`

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

`VGrid`

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Line Plot Showing Multiple Frequencies

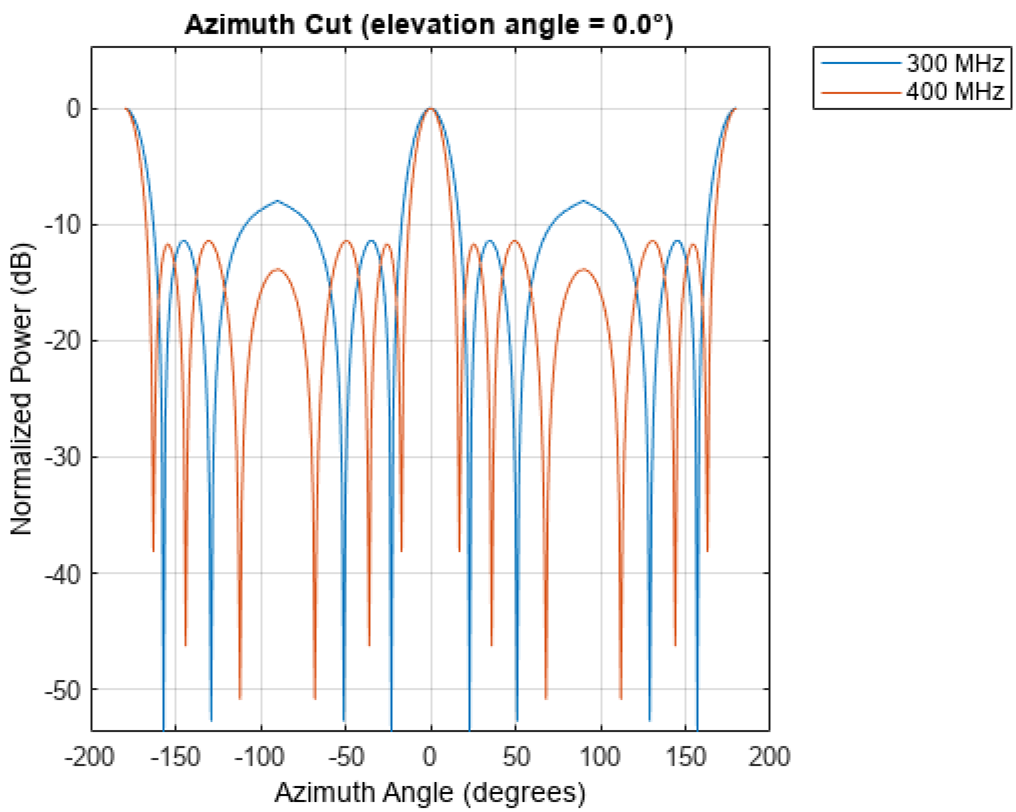
Using a line plot, show the azimuth cut response of a 5-element heterogeneous uniform linear array along 0 degrees elevation. The plot shows the responses at operating frequencies of 200 MHz and 400 MHz.

Construct the array from z-directed and y-directed short dipole antenna elements.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
```

Plot the response.

```
fc = [3e8 4e8];
c = physconst('LightSpeed');
plotResponse(sArray,fc,c);
```



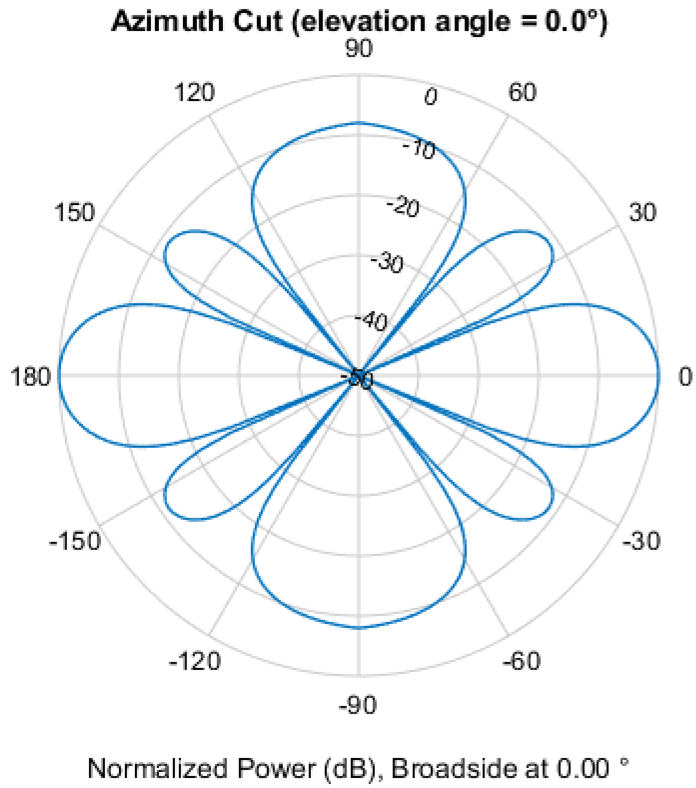
Plot Response and Directivity for 5-Element Array

Construct a 5-element heterogeneous ULA of short-dipole antenna elements. Using the `plotResponse` method, plot the array's azimuth response in polar format. Assume each element's operating frequency spans 200-500 MHz and the wave propagation speed is the speed of light.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
```

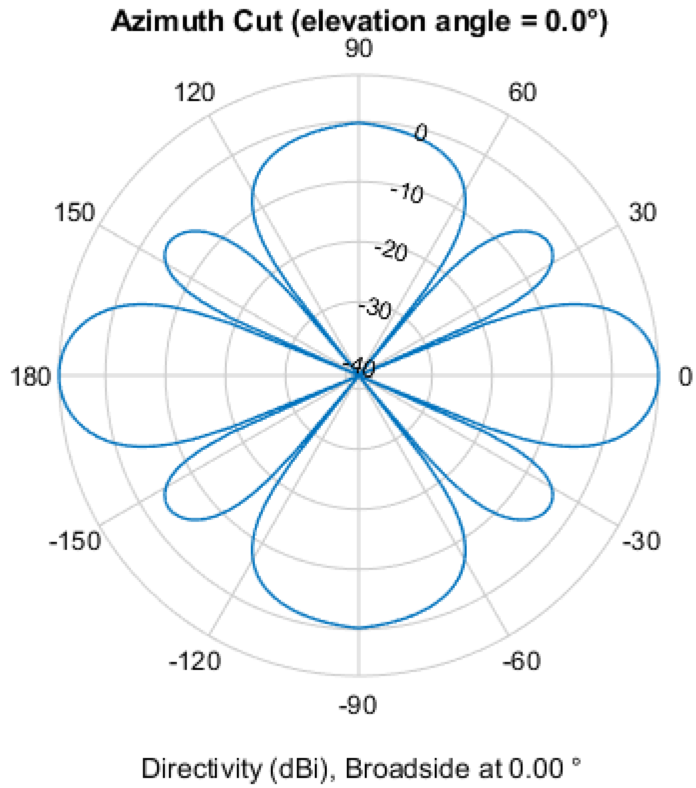
Plot the response at 300 MHz.

```
fc = 3e8;
c = physconst('LightSpeed');
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar');
```



Plot the directivity of the array at 300 MHz.

```
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar',...
    'Unit','dbi');
```



Plot Response for 9-Element Array with Two Weight Sets

Construct a 9-element heterogeneous ULA of short-dipole antenna elements having different orientations. Assume each element response is in the frequency range 200-500 MHz. Using the `plotResponse` method, plot the array's azimuth response in polar format. Use the `Weights` parameter to set two different sets of tapering weights: a uniform tapering and a Taylor tapering. Use the `AzimuthAngles` parameter to restrict the display range from -45 to 45 degrees in 0.1 degree increments.

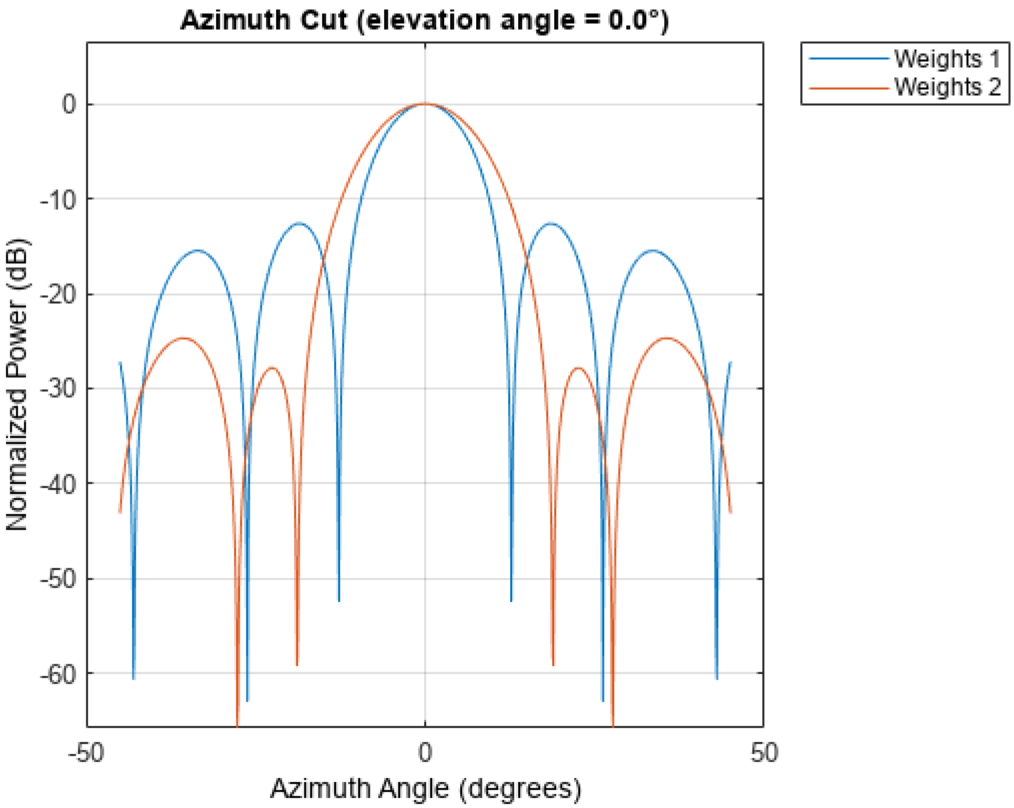
Construct the array.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 2 2 2 2 2 1 1]);
```

Plot the response at 300 MHz.

```
fc = 3e8;
wts1 = ones(9,1);
```

```
wts2 = taylorwin(9);
c = physconst('LightSpeed');
plotResponse(sArray,fc,c,'RespCut','Az',...
    'AzimuthAngles',[-45:0.1:45],...
    'Weights',[wts1,wts2]);
```



As expected, the tapered weighting broadens the mainlobe and reduces the sidelobes.

See Also

uv2azel | azel2uv

step

System object: `phased.HeterogeneousULA`
Package: `phased`

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the array elements' responses `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`H`

Array object

`FREQ`

Operating frequencies of array in hertz. `FREQ` is a row vector of length L. Typical values are within the range specified by a property of `H.Element`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has zero response at frequencies outside that range.

`ANG`

Directions in degrees. `ANG` is either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

If ANG is a row vector of length M, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be 0°.

Output Arguments

RESP

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, RESP, has the dimensions N-by-M-by-L. N is the number of elements in the array. The dimension M is the number of angles specified in ANG. L is the number of frequencies specified in FREQ. For any element, the columns of RESP contain the responses of the array elements for the corresponding direction specified in ANG. Each of the L pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.
- If the array is capable of supporting polarization, the voltage response, RESP, is a MATLAB struct containing two fields, RESP.H and RESP.V. The field, RESP.H, represents the array's horizontal polarization response, while RESP.V represents the array's vertical polarization response. Each field has the dimensions N-by-M-by-L. N is the number of elements in the array, and M is the number of angles specified in ANG. L is the number of frequencies specified in FREQ. Each column of RESP contains the responses of the array elements for the corresponding direction specified in ANG. Each of the L pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.

Examples

Heterogeneous ULA of Cosine Antenna Elements

Create a 5-element heterogeneous ULA of cosine antenna elements with difference responses, and find the response of each element at 30° azimuth.

```
antenna1 = phased.CosineAntennaElement('CosinePower',1.5);
antenna2 = phased.CosineAntennaElement('CosinePower',1.8);
array = phased.HeterogeneousULA(...
    'ElementSet',{antenna1,antenna2},...
    'ElementIndices',[1 2 2 2 1]);
fc = 1e9;
c = physconst('LightSpeed');
ang = [30;0];
resp = array(fc,ang)
```

```
resp = 5×1

    0.8059
    0.7719
    0.7719
    0.7719
    0.8059
```

Response of Heterogeneous Microphone ULA Array

Find the response of a heterogeneous ULA array of 7 custom microphone elements with different responses.

Create two microphones with different response patterns.

```
mic1 = phased.CustomMicrophoneElement(...
    'FrequencyResponse',[20 20e3]);
mic1.PolarPatternFrequencies = [500 1000];
mic1.PolarPattern = mag2db([...
    0.5+0.5*cosd(mic1.PolarPatternAngles);...
    0.6+0.4*cosd(mic1.PolarPatternAngles)]);
mic2 = phased.CustomMicrophoneElement(...
    'FrequencyResponse',[20 20e3]);
mic2.PolarPatternFrequencies = [500 1000];
mic2.PolarPattern = mag2db([...
    ones(size(mic2.PolarPatternAngles));...
    ones(size(mic2.PolarPatternAngles))]);
```

Create the heterogeneous ULA.

```
array = phased.HeterogeneousULA(...
    'ElementSet',{mic1,mic2},...
    'ElementIndices',[1 1 2 2 2 1 1]);
```

Find the array response at 40° and 50° azimuth.

```
fc = [1500, 2000];
ang = [40 50; 0 0];
resp = array(fc,ang)
```

```
resp = 
resp(:,:,1) =

    9.0642    8.5712
    9.0642    8.5712
   10.0000   10.0000
   10.0000   10.0000
   10.0000   10.0000
    9.0642    8.5712
    9.0642    8.5712


resp(:,:,2) =

    9.0642    8.5712
    9.0642    8.5712
   10.0000   10.0000
   10.0000   10.0000
   10.0000   10.0000
    9.0642    8.5712
    9.0642    8.5712
```

See Also

uv2azel | phitheta2azel

viewArray

System object: `phased.HeterogeneousULA`
Package: `phased`

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

`viewArray(H)` plots the geometry of the array specified in `H`.

`viewArray(H,Name,Value)` plots the geometry of the array, with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = viewArray( ___ )` returns the handle of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

`H`

Array object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Parent`

Handle to the axes along which the array geometry is displayed.

`ShowNormals`

Set this value to `true` to show the normal directions of all elements of the array. Set this value to `false` to plot the elements without showing normal directions.

Default: `false`

`ShowLocalCoordinates`

Logical flag specifying whether to show the local coordinate axes.

Default: `true`

`ShowAnnotation`

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Default: `true`

`Orientation`

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Default: `[0;0;0]`

`ShowTaper`

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

`ShowIndex`

Vector specifying the element indices to show in the figure. Each number in the vector must be an integer between 1 and the number of elements. You can also specify the value `'All'` to show the indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

`Title`

Character vector specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

`hPlot`

Handle of array elements in figure window.

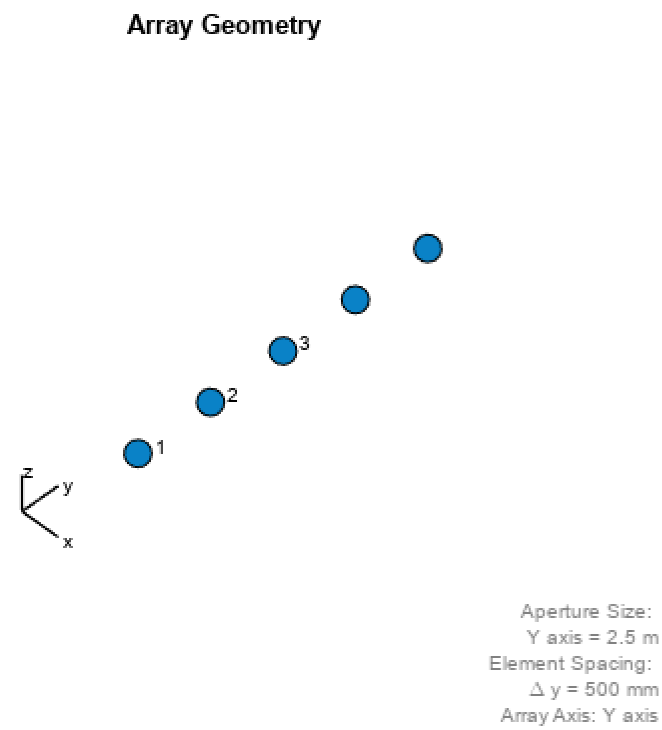
Examples

Geometry and Indices of Heterogeneous ULA Elements

Display the geometry of a 5-element heterogeneous ULA of cosine antenna elements, showing the indices for the first three elements.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
```

```
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
viewArray(sArray,'ShowIndex',[1:3])
```



See Also
`phased.ArrayResponse`

Topics
Phased Array Gallery

phased.HeterogeneousURA

Package: `phased`

Heterogeneous uniform rectangular array

Description

The `HeterogeneousURA` object constructs a heterogeneous uniform rectangular array (URA).

To compute the response for each element in the array for specified directions:

1. Define and set up your uniform rectangular array. See "Construction" on page 1-613.
2. Call `step` to compute the response according to the properties of `phased.HeterogeneousURA`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.HeterogeneousURA` creates a heterogeneous uniform rectangular array (URA) System object, `H`. This object models a heterogeneous URA formed with sensor elements whose pattern may vary from element to element. Array elements are distributed in the *yz*-plane in a rectangular lattice. An *M*-by-*N* heterogeneous URA has *M* rows and *N* columns. The array boresight direction is along the positive *x*-axis. The default array is a 2-by-2 URA of isotropic antenna elements.

`H = phased.HeterogeneousURA(Name,Value)` creates the object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`ElementSet`

Set of elements used in the array

Set of elements used in the sensor array, specified as a row MATLAB cell array. Elements specified in the `ElementSet` property must be either Phased Array System Toolbox antennas, microphones, or transducers System objects or Antenna Toolbox System object. In addition, all specified antenna elements must have the same polarization capability.

Default: One cell containing one isotropic antenna element

`ElementIndices`

Elements location assignment

This property specifies the mapping of elements in the array. The property assigns elements to their locations in the array using the indices into the `ElementSet` property. The value of

`ElementIndices` must be an *M*-by-*N* matrix. In this matrix, *M* represents the number of rows and *N* represents the number of columns. Rows are along *y*-axis and columns are along *z*-axis of the local coordinate system. The values in the matrix specified by `ElementIndices` should be less than or equal to the number of entries in the `ElementSet` property.

Default: `[1 1;1 1]`

`ElementSpacing`

Element spacing

A 1-by-2 vector or a scalar containing the element spacing (in meters) of the array. If `ElementSpacing` is a 1-by-2 vector, it is in the form of `[SpacingBetweenRows,SpacingBetweenColumns]`. See “Spacing Between Columns” on page 1-618 and “Spacing Between Rows” on page 1-618. If `ElementSpacing` is a scalar, both spacings are the same.

Default: `[0.5 0.5]`

`Lattice`

Element lattice

Specify the element lattice as one of `'Rectangular' | 'Triangular'`. When you set the Lattice property to `'Rectangular'`, all elements in the heterogeneous URA are aligned in both row and column directions. When you set the `Lattice` property to `'Triangular'`, the elements in even rows are shifted toward the positive row axis direction by a distance of half the element spacing along the row.

Default: `'Rectangular'`

`ArrayNormal`

Array normal direction

Array normal direction, specified as one of `'x'`, `'y'`, or `'z'`.

URA elements lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction

ArrayNormal Property Value	Element Positions and Boresight Directions
`'x'`	Array elements lie on the *yz*-plane. All element boresight vectors point along the *x*-axis.
`'y'`	Array elements lie on the *zx*-plane. All element boresight vectors point along the *y*-axis.
`'z'`	Array elements lie on the *xy*-plane. All element boresight vectors point along the *z*-axis.

Default: `'x'`

`Taper`

Element tapers

Element tapers, specified as a complex-valued scalar, or a complex-valued 1-by-*MN* row vector, *MN*-by-1 column vector, or *M*-by-*N* matrix. Tapers are applied to each element in the sensor array. Tapers are often referred to as element weights. *M* is the number of elements along the *z*-axis, and *N* is the number of elements along *y*-axis. *M* and *N* correspond to the values of `[NumberofRows, NumberOfColumns]` in the `Size` property. If `Taper` is a scalar, the same taper value is applied to all elements. If the value of `Taper` is a vector or matrix, taper values are applied to the corresponding elements. Tapers are used to modify both the amplitude and phase of the received data.

Default: 1

Methods

Specific to `phased.HeterogeneousURA` Object	
`beamwidth`	Compute and display beamwidth of an array
`collectPlaneWave`	Simulate received plane waves
`directivity`	Directivity of heterogeneous uniform rectangular array
`getElementNormal`	Normal vector to array elements
`getElementPosition`	Positions of array elements
`getNumElements`	Number of elements in array
`getTaper`	Array element tapers
`isPolarizationCapable`	Polarization capability
`pattern`	Plot heterogeneous URA directivity and power pattern
`patternAzimuth`	Plot heterogeneous URA directivity or pattern versus azimuth
`patternElevation`	Plot heterogeneous URA array directivity or pattern versus elevation
`perturbations`	Perturbations defined on phased array
`perturbedArray`	Apply perturbations to phased array
`perturbedPattern`	Compute and plot azimuth pattern of perturbed array
`step`	Output responses of array elements
`viewArray`	View array geometry

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Azimuth Pattern of 3-by-2 Heterogeneous URA

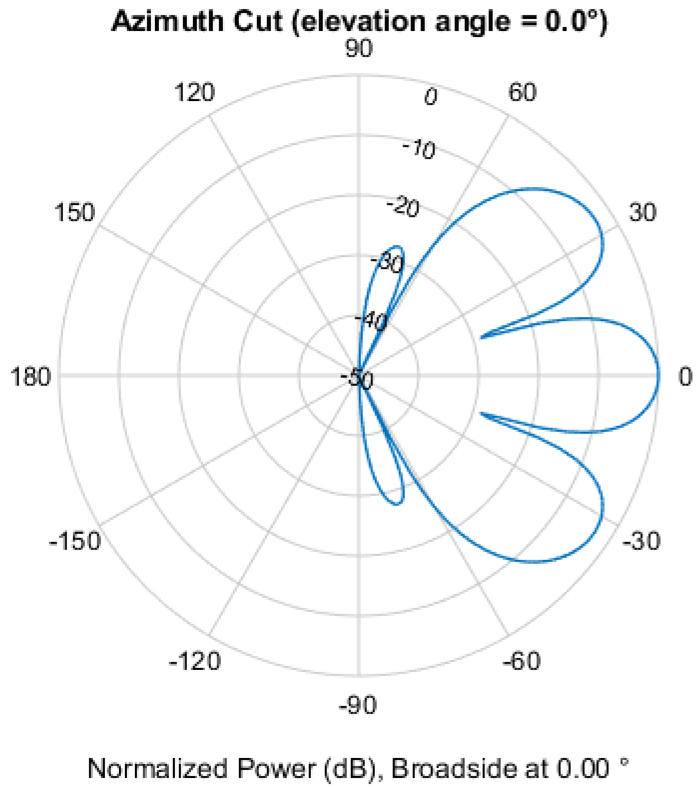
Construct a 3-by-2 heterogeneous URA with a rectangular lattice, and find the response of each element at 30 degrees azimuth and 0 degrees elevation. Assume the operating frequency is 1 GHz.

```
antenna1 = phased.CosineAntennaElement('CosinePower',1.5);
antenna2 = phased.CosineAntennaElement('CosinePower',1.8);
array = phased.HeterogeneousURA('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 1; 2 2; 1 1]);
fc = 1e9;
ang = [30;0];
resp = array(fc,ang)
```

```
resp = 6×1

    0.8059
    0.7719
    0.8059
    0.8059
    0.7719
    0.8059
```

Plot the azimuth pattern of the array.

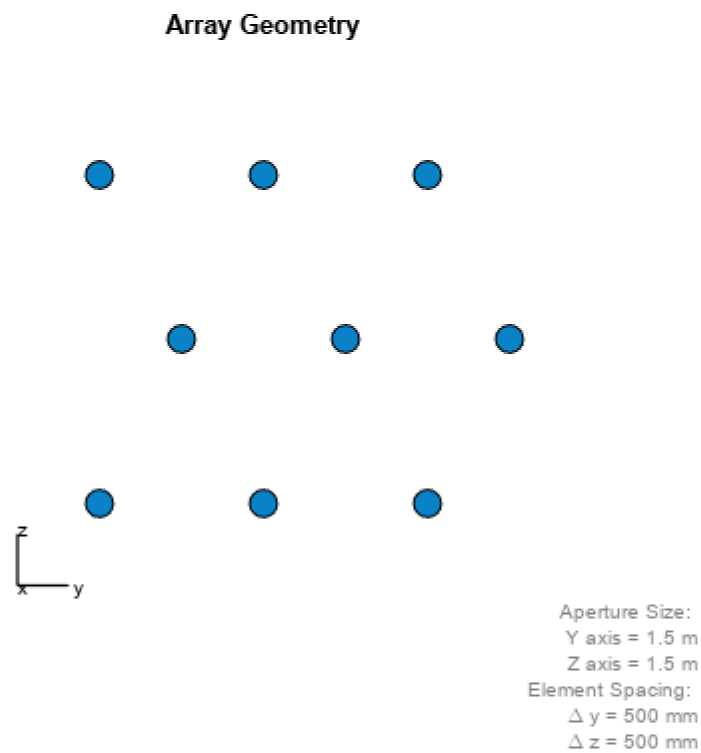
```
c = physconst('LightSpeed');
pattern(array,fc,[-180:180],0,'PropagationSpeed',c, ...
    'CoordinateSystem','polar','Type','powerdb','Normalize',true)
```



Draw Heterogeneous Triangular Lattice Array

Construct a 3-by-3 heterogeneous URA with a triangular lattice. The element spacing is 0.5 meter. Display the array shape.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1],...
    'Lattice','Triangular');
viewArray(sArray);
```



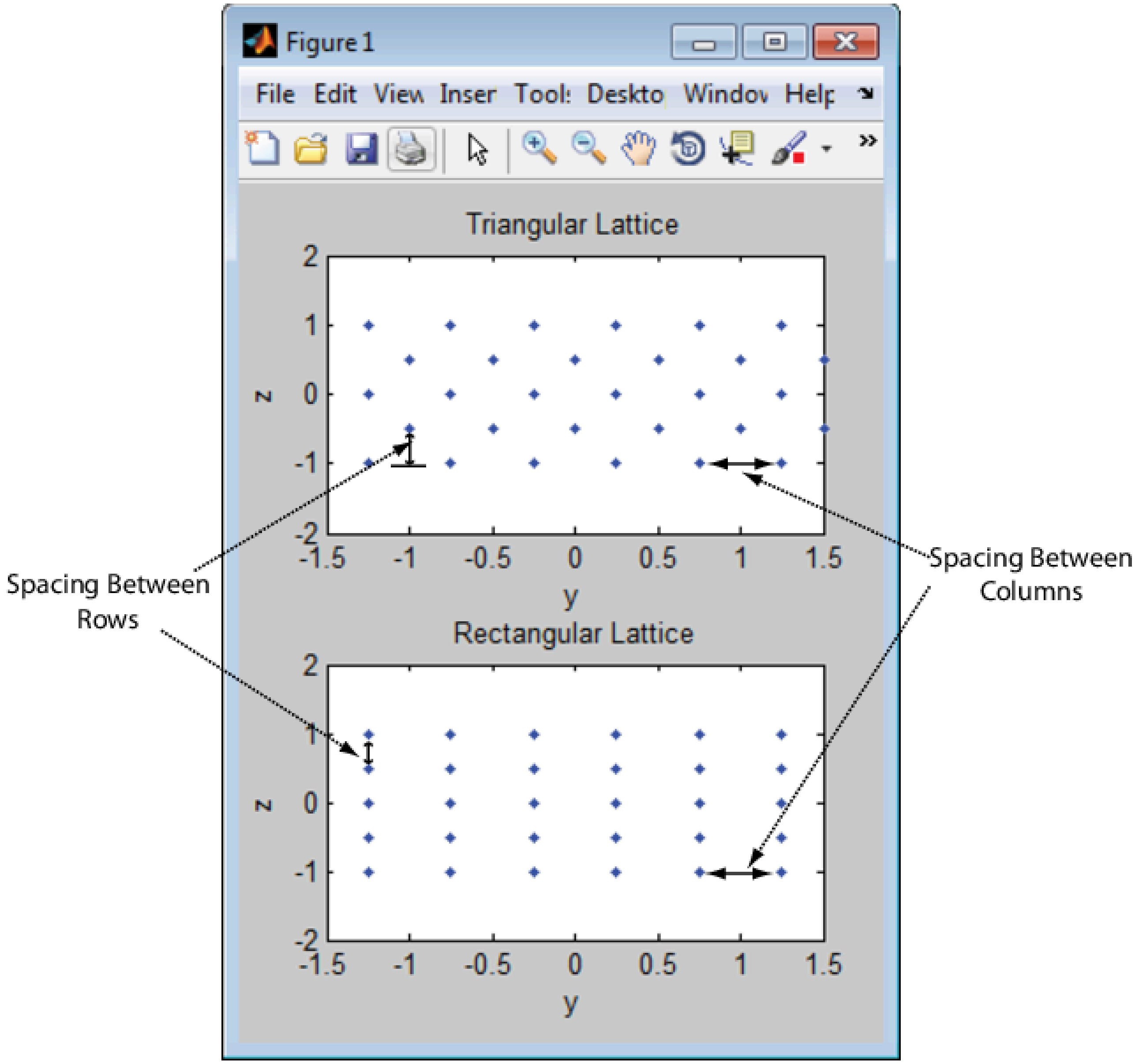
More About

Spacing Between Columns

The spacing between columns is the distance between adjacent elements in the same row.

Spacing Between Rows

The spacing between rows is the distance along the column axis direction between adjacent rows.



Version History

Introduced in R2013a

References

[1] Brookner, E., ed. *Radar Technology*. Lexington, MA: LexBook, 1996.

[2] Brookner, E., ed. *Practical Phased Array Antenna Systems*. Boston: Artech House, 1991.

[3] Mailloux, R. J. “Phased Array Theory and Technology,” *Proceedings of the IEEE*, Vol., 70, Number 3, 1982, pp. 246–291.

[4] Mott, H. *Antennas for Radar and Communications, A Polarimetric Approach*. New York: John Wiley & Sons, 1992.

[5] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, `plotResponse`, and `viewArray` methods are not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also
`phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.URA` | `phased.HeterogeneousULA` | `phased.UCA` | `phased.HeterogeneousConformalArray`

Topics
Phased Array Gallery

directivity

System object: `phased.HeterogeneousURA`
Package: `phased`

Directivity of heterogeneous uniform rectangular array

Syntax

```
D = directivity(H,FREQ,ANGLE)
D = directivity(H,FREQ,ANGLE,Name,Value)
```

Description

`D = directivity(H,FREQ,ANGLE)` computes the "Directivity (dBi)" on page 1-624 of a heterogeneous uniform rectangular array of antenna or microphone elements, `H`, at frequencies specified by the `FREQ` and in angles of direction specified by the `ANGLE`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`D = directivity(H,FREQ,ANGLE,Name,Value)` computes the directivity with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

H — Heterogeneous uniform rectangular array

System object

Uniform rectangular array specified as a `phased.HeterogeneousURA` System object.

Example: `H = phased.HeterogeneousURA`

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

ANGLE — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

PropagationSpeed — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Heterogeneous Uniform Rectangular Array

Compute the directivity of a 9-element 3-by-3 heterogeneous URA consisting of short-dipole antenna elements. The three elements on the middle row are Y-directed while all the remaining elements are Z-directed.

Set the signal frequency to 1 GHz.

```
c = physconst('LightSpeed');
freq = 1e9;
lambda = c/freq;
```

Create the array of short-dipole antenna elements. The elements have frequency ranges from 0 to 10 GHz.

```
myElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[0 10e9],...
    'AxisDirection','Z');
myElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[0 10e9],...
    'AxisDirection','Y');
myArray = phased.HeterogeneousURA(...
    'ElementSet',{myElement1,myElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1]);
```

Create the steering vector to point to 30 degrees azimuth and compute the directivity in the same direction as the steering vector.

```
ang = [30;0];
w = steervec(getElementPosition(myArray)/lambda,ang);
```

```
d = directivity(myArray,freq,ang,'PropagationSpeed',c,...
    'Weights',w)
```

```
d = 11.1405
```

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternAzimuth` | `patternElevation`

collectPlaneWave

System object: `phased.HeterogeneousURA`
Package: `phased`

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

`Y = collectPlaneWave(H,X,ANG)` returns the received signals at the sensor array, `H`, when the input signals indicated by `X` arrive at the array from the directions specified in `ANG`.

`Y = collectPlaneWave(H,X,ANG,FREQ)`, in addition, specifies the incoming signal carrier frequency in `FREQ`.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)`, in addition, specifies the signal propagation speed in `C`.

Input Arguments

H

Array object.

X

Incoming signals, specified as an M-column matrix. Each column of `X` represents an individual incoming signal.

ANG

Directions from which incoming signals arrive, in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column specifies the direction of arrival of the corresponding signal in `X`. Each column of `ANG` is in the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `ANG` is a row vector of length M, each entry in `ANG` specifies the azimuth angle. In this case, the corresponding elevation angle is assumed to be 0°.

FREQ

Carrier frequency of signal in hertz. `FREQ` must be a scalar.

Default: `3e8`

C

Propagation speed of signal in meters per second.

Default: Speed of light

Output Arguments

Y

Received signals. Y is an N-column matrix, where N is the number of elements in the array H. Each column of Y is the received signal at the corresponding array element, with all incoming signals combined.

Examples

Create Received Signal at Heterogeneous URA

Simulate two received signals at a 2-by-2 element heterogeneous URA with two different cosine antenna patterns. The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0°.

```
antenna1 = phased.CosineAntennaElement('CosinePower',1.5);
antenna2 = phased.CosineAntennaElement('CosinePower',1.8);
array = phased.HeterogeneousURA(...
    'ElementSet',{antenna1,antenna2},...
    'ElementIndices',[1 2; 1 2]);
y = collectPlaneWave(array,randn(4,2),[10 30],1e8,physconst('LightSpeed'))
```

```
y = 4×4 complex

   0.8433 - 0.1314i   0.8433 - 0.1314i   0.8433 + 0.1314i   0.8433 + 0.1314i
   0.5632 + 0.1721i   0.5632 + 0.1721i   0.5632 - 0.1721i   0.5632 - 0.1721i
  -2.6683 + 0.3175i  -2.6683 + 0.3175i  -2.6683 - 0.3175i  -2.6683 - 0.3175i
   1.1895 - 0.1671i   1.1895 - 0.1671i   1.1895 + 0.1671i   1.1895 + 0.1671i
```

Algorithms

`collectPlaneWave` modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. This method does not account for the response of individual elements in the array.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

getElementNormal

System object: `phased.HeterogeneousURA`
Package: `phased`

Normal vector to array elements

Syntax

```
normvec = getElementNormal(sURA)
normvec = getElementNormal(sURA,elemidx)
```

Description

`normvec = getElementNormal(sURA)` returns the normal vectors of the array elements of the `phased.URA` System object, `sURA`. The output argument `normvec` is a 2-by-*N* matrix, where *N* is the number of elements in array, `sURA`. Each column of `normvec` defines the normal direction of an element in the local coordinate system in the form`[az;el]`. Units are degrees. The origin of the local coordinate system is defined by the phase center of the array.

`normvec = getElementNormal(sURA,elemidx)` returns only the normal vectors of the elements specified in the element index vector, `elemidx`. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

`sURA` — Heterogeneous uniform rectangular array
`phased.HeterogeneousURA` System object

Uniform line array, specified as a `phased.HeterogeneousURA` System object.

Example: `sULA = phased.HeterogeneousURA`

`elemidx` — Element indices
all array elements (default) | integer-valued 1-by-*M* row vector | integer-valued *M*-by-1 column vector

Element indices , specified as a 1-by-*M* or *M*-by-1 vector. Index values lie in the range 1 to *N* where *N* is the number of elements of the array. When `elemidx` is specified, `getElementNormal` returns the normal vectors of the elements contained in `elemidx`.

Example: `[1,5,4]`

Output Arguments

`normvec` — Element normal vectors
2-by-*P* real-valued vector

Element normal vectors, specified as a 2-by-*P* real-valued vector. Each column of `normvec` takes the form `[az,el]`. When `elemidx` is not specified, *P* equals the array dimension. When `elemidx` is specified, *P* equals the length of `elemidx`, *M*. You can determine element indices using the `viewArray` method.

Examples

URA Element Normals

Construct three 2-by-2 URA's with element normals along the *x*-, *y*-, and *z*-axes. Obtain the element positions and normal directions.

First, choose the array normal along the *x*-axis.

```
sURA1 = phased.URA('Size',[2,2],'ArrayNormal','x');
pos = getElementPosition(sURA1)
```

```
pos = 3×4

         0         0         0         0
   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
```

```
normvec = getElementNormal(sURA1)
```

```
normvec = 2×4

     0     0     0     0
     0     0     0     0
```

All elements lie in the *yz*-plane and the element normal vectors point along the *x*-axis *(0°,0°)*.

Next, choose the array normal along the *y*-axis.

```
sURA2 = phased.URA('Size',[2,2],'ArrayNormal','y');
pos = getElementPosition(sURA2)
```

```
pos = 3×4

    0.2500    0.2500   -0.2500   -0.2500
         0         0         0         0
    0.2500   -0.2500    0.2500   -0.2500
```

```
normvec = getElementNormal(sURA2)
```

```
normvec = 2×4

    90    90    90    90
     0     0     0     0
```

All elements lie in the *zx*-plane and the element normal vectors point along the *y*-axis *(90°,0°)*.

Finally, set the array normal along the *z*-axis. Obtain the normal vectors of the odd-numbered elements.

```
sURA3 = phased.URA('Size',[2,2],'ArrayNormal','z');
pos = getElementPosition(sURA3)
```

```
pos = 3×4
```

```
   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
         0         0         0         0
```

```
normvec = getElementNormal(sURA3,[1,3])
```

```
normvec = 2×2

     0     0
    90    90
```

All elements lie in the *xy*-plane and the element normal vectors point along the *z*-axis *(0°,90°)*.

Version History

Introduced in R2016a

getElementPosition

System object: phased.HeterogeneousURA
Package: phased

Positions of array elements

Syntax

```
POS = getElementPosition(H)
POS = getElementPosition(H,ELEIDX)
```

Description

`POS = getElementPosition(H)` returns the element positions of the HeterogeneousURA System object, `H`. `POS` is a 3-by-N matrix where N is the number of elements in `H`. Each column of `POS` defines the position of an element in the local coordinate system, in meters, using the form `[x; y; z]`.

For details regarding the local coordinate system of the URA or heterogeneous URA, enter `phased.URA.coordinateSystemInfo` on the command line.

`POS = getElementPosition(H,ELEIDX)` returns the positions of the elements that are specified in the element index vector, `ELEIDX`. The element indices of a URA run down each column, then to the top of the next column to the right. For example, in a URA with 4 elements in each row and 3 elements in each column, the element in the third row and second column has an index value of 6. This syntax can use any of the input arguments in the previous syntax.

Examples

Element Positions of Heterogeneous URA

Construct a heterogeneous URA with a rectangular lattice, and obtain the element positions.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
array = phased.HeterogeneousURA('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 2; 2 1]);
pos = getElementPosition(array)
```

```
pos = 3×4

         0         0         0         0
   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
```

getNumElements

System object: phased.HeterogeneousURA
Package: phased

Number of elements in array

Syntax

N = getNumElements(H)

Description

N = getNumElements(H) returns the number of elements, N, in the HeterogeneousURA System object H.

Examples

Find Number of Elements of Heterogeneous URA

Construct a Heterogeneous URA, and obtain the number of elements.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
array = phased.HeterogeneousURA('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 2; 2 1]);
N = getNumElements(array)
```

```
N = 4
```

getTaper

System object: `phased.HeterogeneousURA`
Package: `phased`

Array element tapers

Syntax

`wts = getTaper(h)`

Description

`wts = getTaper(h)` returns the tapers, `wts`, applied to each element of the phased heterogeneous uniform rectangular array (URA), `h`. Tapers are often referred to as weights.

Input Arguments

h — Uniform rectangular array
`phased.HeterogeneousURA` System object

Uniform rectangular array specified as a `phased.HeterogeneousURA` System object.

Output Arguments

wts — Array element tapers
N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1, complex-valued vector. The dimension *N* is the number of elements in the array. The array tapers are returned in the same order as the element indices. The element indices of a URA run down each column, then to the top of the next column to the right.

Examples

Heterogeneous URA Array Element Tapering

Construct a 2-by-5 element heterogeneous URA with a Taylor window taper along each row. Then, show the array with the element taper shading.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Y');
array = phased.HeterogeneousURA('ElementSet',{antenna1,antenna2},...
    'ElementIndices',[1 2 2 2 1 ; 1 2 2 2 1],...
    'Taper',[taylorwin(5)';taylorwin(5)']);
w = getTaper(array)
```

```
w = 10×1
```

```
0.5181
0.5181
1.2029
1.2029
1.5581
1.5581
1.2029
1.2029
0.5181
0.5181
```

isPolarizationCapable

System object: `phased.HeterogeneousURA`
Package: `phased`

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h — Uniform rectangular array

Uniform rectangular array specified as `phased.HeterogeneousURA` System object.

Output Arguments

`flag` — Polarization-capability flag

Polarization-capability flag returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

Short-Dipole Antenna Array Polarization

Show that an array of short-dipole antenna element supports polarization.

```
antenna1 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9], ...
    'AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
array = phased.HeterogeneousURA('ElementSet',{antenna1,antenna2}, ...
    'ElementIndices',[1 2 2 2 1 ; 1 2 2 2 1]);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

pattern

System object: `phased.HeterogeneousURA`
Package: `phased`

Plot heterogeneous URA directivity and power pattern

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sArray,FREQ)` plots the 3-D array directivity pattern (in dBi) for the array specified in `sArray`. The operating frequency is specified in `FREQ`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`pattern(sArray,FREQ,AZ)` plots the array directivity pattern at the specified azimuth angle.

`pattern(sArray,FREQ,AZ,EL)` plots the array directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the array pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See "Convert plotResponse to pattern" on page 1-645 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sArray` — Heterogeneous URA
System object

Heterogeneous conformal array, specified as a `phased.HeterogeneousURA` System object.

Example: `sArray= phased.HeterogeneousURA;`

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Orientation` — Array orientation
`[0;0;0]`. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

`Normalize` — Display normalize pattern
`true` (default) | `false`

Display normalized pattern, specified as the comma-separated pair consisting of `'Normalize`' and a Boolean. Set this parameter to `true` to display a normalized pattern. This parameter does not apply when you set `'Type'` to `'directivity'`. Directivity patterns are already normalized.

Data Types: `logical`

`ShowArray` — View array geometry
`false` (default) | `true`

View the array geometry along with the 3D radiation pattern, specified as `false` or `true`.

Data Types: `logical`

`ShowLocalCoordinates` — Show local coordinate axes
`true` (default) | `false`

Show the local coordinate axes, specified as `true` or `false`.

Data Types: `logical`

`ShowColorbar` — Show colorbar
`true` (default) | `false`

Show the colorbar, specified as `true` or `false`.

Data Types: `logical`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

PAT — Array pattern
M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of `PAT` correspond to the dimensions of the output arguments `AZ_ANG` and `EL_ANG`.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in `AZ`. The columns of `PAT` correspond to the values in `AZ_ANG`. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

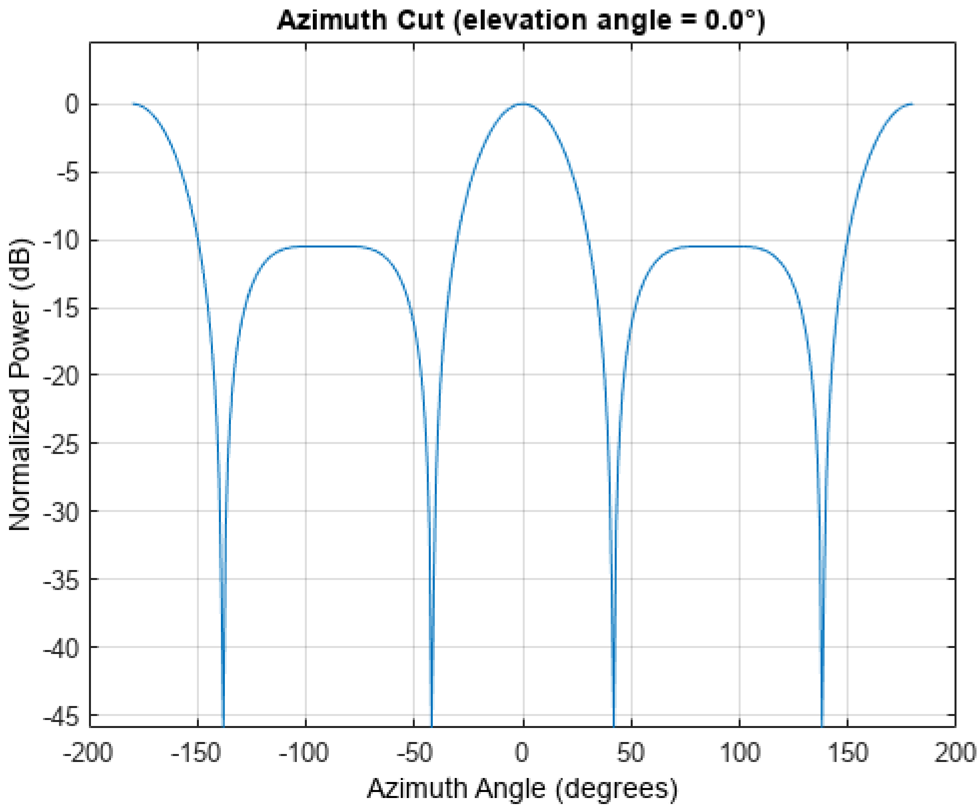
Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

Examples

Azimuth Pattern and Directivity of Heterogeneous URA

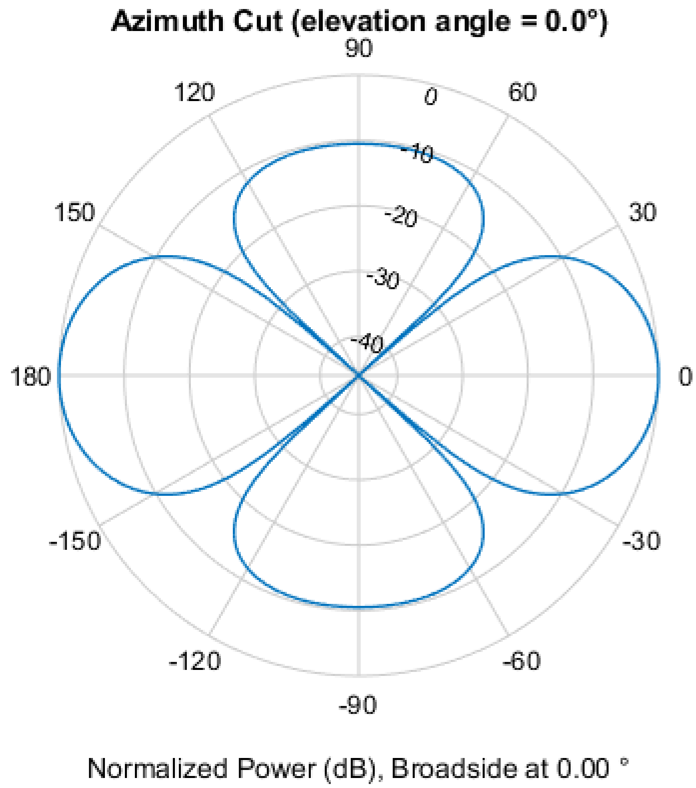
Construct a 3-by-3 heterogeneous URA of short-dipole antenna elements with a rectangular lattice. Then, plot the array's azimuth pattern at 300 MHz.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1]);
fc = 300e6;
c = physconst('LightSpeed');
pattern(sArray,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','rectangular',...
    'Type','powerdb',...
    'Normalize',true,...
    'Polarization','combined')
```



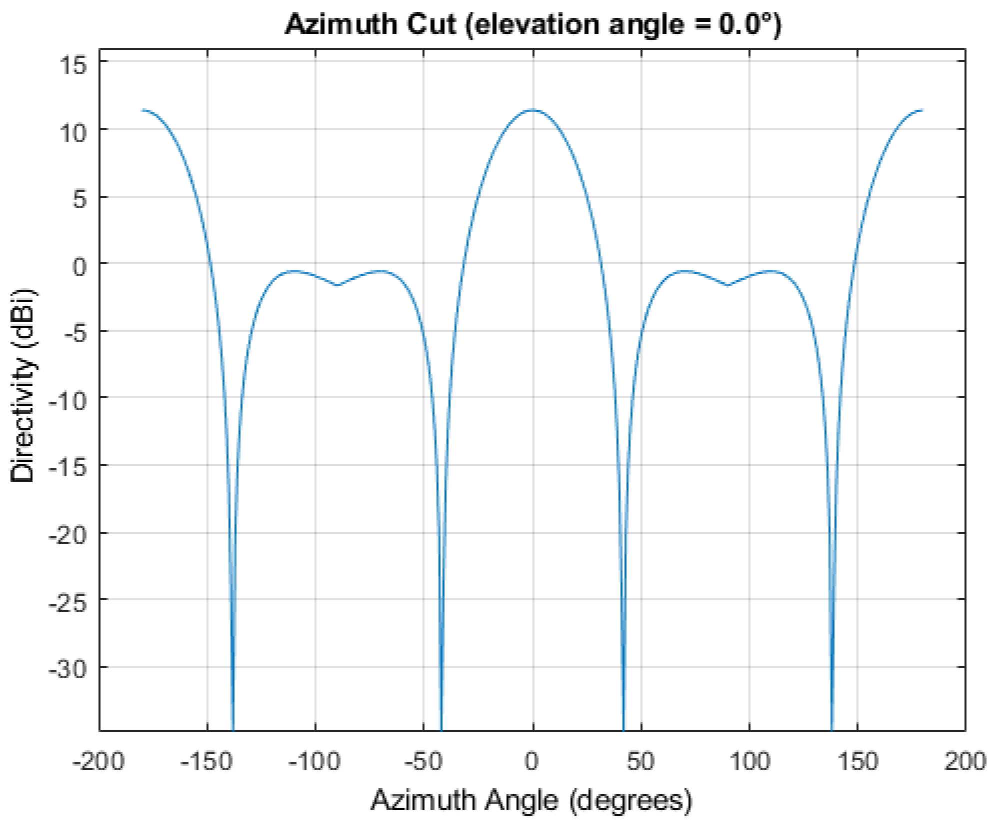
Plot the same result in polar form.

```
pattern(sArray,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Normalize',true,...
    'Polarization','combined')
```



Finally, plot the directivity.

```
pattern(sArray,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','rectangular',...
    'Type','directivity')
```

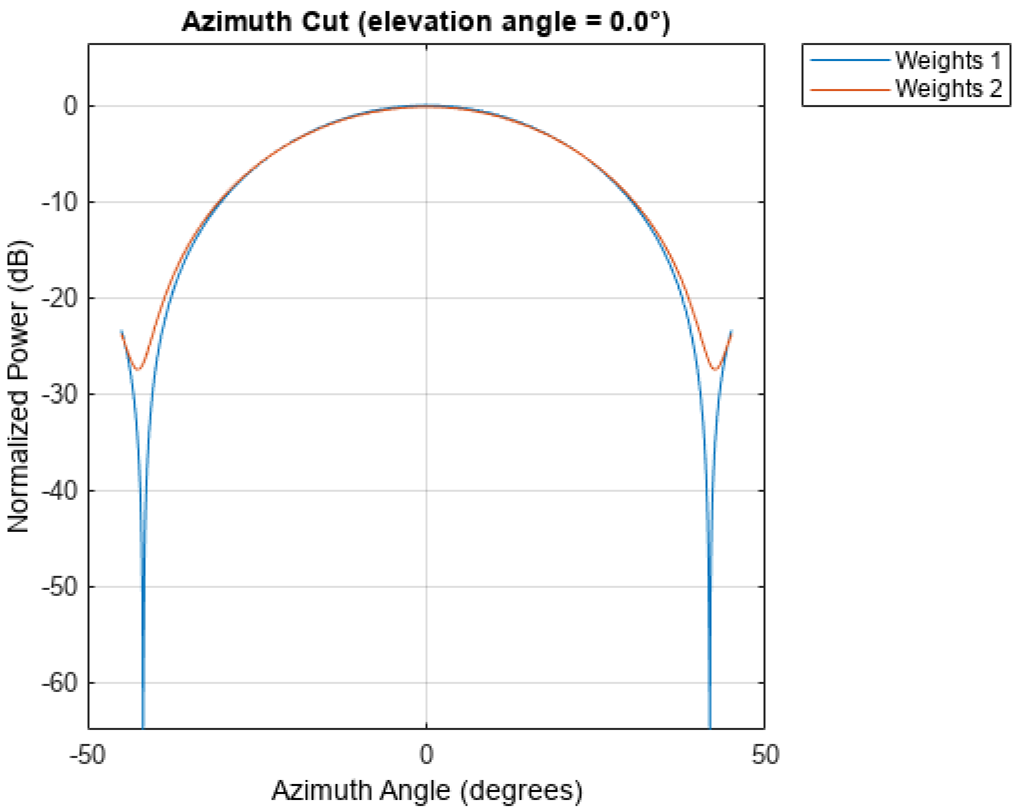


Azimuth Pattern of Heterogeneous URA for Two Sets of Weights

Construct a square 3-by-3 heterogeneous URA composed of 9 short-dipole antenna elements with different orientations. Plot the array azimuth pattern from -45 degrees to 45 degrees in 0.1 degree increments. The `Weights` parameter lets you display the array pattern simultaneously for different sets of weights: in this case a uniform set of weights and a tapered set.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1]);
fc = [3e8];
c = physconst('LightSpeed');
wts1 = ones(9,1)/9;
wts2 = [.7,.7,.7,.7,1,.7,.7,.7,.7]';
wts2 = wts2/sum(wts2);
pattern(sArray,fc,[-45:0.1:45],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','rectangular',...
```

```
'Type','powerdb',...
'Weights',[wts1,wts2],...
'Polarization','combined')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or *UV* space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.

Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.

Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>*UV* space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>*UV* space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.

'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>*UV* space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a *U*-space vector. Use EL to specify a *V*-space scalar.</td></tr>
<tr><td>*UV* space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a *U*-space vector. Use EL to specify a *V*-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the *UV* grid values using AZ and EL.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td><code>'UV'</code>. Set the display range using both the <code>'UGrid'</code> and <code>'VGrid'</code> name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td><code>'CutAngle'</code> name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when <code>'RespCut'</code> is set to <code>'Az'</code> or <code>'El'</code>, use <code>'CutAngle'</code> to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either <code>AZ</code> or <code>EL</code> as a scalar, not a vector.</td></tr>
<tr><td><code>'NormalizeResponse'</code> name-value pair</td><td>Normalizes the plot. When <code>'Unit'</code> is set to <code>'dbi'</code>, you cannot specify <code>'NormalizeResponse'</code>.</td><td>Use the <code>'Normalize'</code> name-value pair. When <code>'Type'</code> is set to <code>'directivity'</code> you cannot specify <code>'Normalize'</code>.</td></tr>
<tr><td><code>'OverlayFreq'</code> name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when <code>'Format'</code> is set to <code>'line'</code> or <code>'uv'</code> and <code>'RespCut'</code> is not set to <code>'3D'</code>. The value <code>true</code> produces an overlay plot and the value <code>false</code> produces a waterfall plot.</td><td><code>'PlotStyle'</code> name-value pair plots multiple frequencies on the same 2-D plot.

The values <code>'overlay'</code> and <code>'waterfall'</code> correspond to <code>'OverlayFreq'</code> values of <code>true</code> and <code>false</code>. The option <code>'waterfall'</code> is allowed only when <code>'CoordinateSystem'</code> is set to <code>'rectangular'</code> or <code>'uv'</code>.</td></tr>
<tr><td><code>'Polarization'</code> name-value pair</td><td>Determines how to plot polarized fields. Options are <code>'None'</code>, <code>'Combined'</code>, <code>'H'</code>, or <code>'V'</code>.</td><td><code>'Polarization'</code> name-value pair determines how to plot polarized fields. The <code>'None'</code> option is removed. The options <code>'Combined'</code>, <code>'H'</code>, or <code>'V'</code> are unchanged.</td></tr>
<tr><td><code>'Unit'</code> name-value pair</td><td>Determines the plot units. Choose <code>'db'</code>, <code>'mag'</code>, <code>'pow'</code>, or <code>'dbi'</code>, where the default is <code>'db'</code>.</td><td><code>'Type'</code> name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td><code>'db'</code></td><td><code>'powerdb'</code></td></tr><tr><td><code>'mag'</code></td><td><code>'efield'</code></td></tr><tr><td><code>'pow'</code></td><td><code>'power'</code></td></tr><tr><td><code>'dbi'</code></td><td><code>'directivity'</code></td></tr></table></td></tr>
<tr><td><code>'Weights'</code> name-value pair</td><td>Array element tapers (or weights).</td><td><code>'Weights'</code> name-value pair (no change).</td></tr>
<tr><td><code>'AzimuthAngles'</code> name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td><code>AZ</code> argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `patternElevation`

patternAzimuth

System object: `phased.HeterogeneousURA`
Package: `phased`

Plot heterogeneous URA directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sArray,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(sArray,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sArray,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sArray` — Heterogeneous URA

System object

Heterogeneous URA, specified as a `phased.HeterogeneousURA` System object.

Example: `sArray= phased.HeterogeneousURA;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

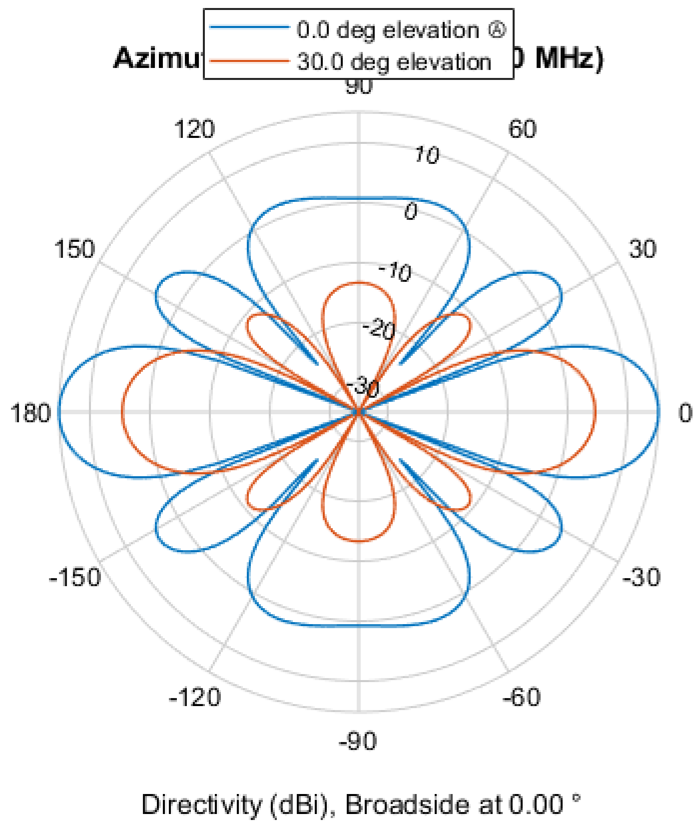
Examples

Azimuth Directivity of Heterogeneous URA

Construct a square 4-by-4 heterogeneous URA composed of a mix of crossed-dipole and short-dipole antenna elements with short dipoles in the center. Plot the array azimuth directivity for two different elevation angles. Set the operating frequency to 400 MHz.

```
sElement1 = phased.CrossedDipoleAntennaElement(...
    'FrequencyRange',[200e6 500e6]);
```

```
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[200e6 500e6],...
    'AxisDirection','Z');
elemindices = ones(4,4);
elemindices(2:3,2:3) = 2;
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',elemindices);
fc = 400e6;
c = physconst('LightSpeed');
patternAzimuth(sArray,fc,[0 30],...
    'PropagationSpeed',c,...
    'Type','directivity')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: `phased.HeterogeneousURA`
Package: `phased`

Plot heterogeneous ULA directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sArray,FREQ)` plots the 2-D array directivity pattern versus elevation (in dBi) for the array `sArray` at zero degrees azimuth angle. When `AZ` is a vector, multiple overlaid plots are created. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternElevation(sArray,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sArray,FREQ,AZ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sArray` — Heterogeneous URA

System object

Heterogeneous URA array, specified as a `phased.HeterogeneousURA` System object.

Example: `sArray= phased.HeterogeneousURA;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

`AZ` — Azimuth angles for computing directivity and pattern

1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Type` — Displayed pattern type

`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed

speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

`PAT` — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

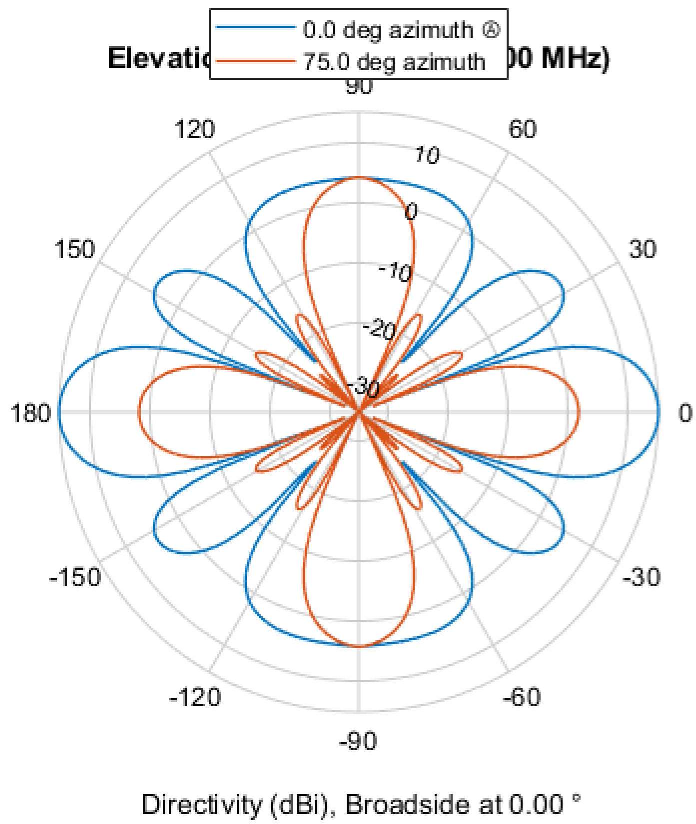
Examples

Elevation Directivity of Heterogeneous URA

Construct a square 4-by-4 heterogeneous URA composed of a mix of crossed-dipole and short-dipole antenna elements with short dipoles in the center. Plot the array elevation directivity for two different azimuth angles. Set the operating frequency to 400 MHz.

```
sElement1 = phased.CrossedDipoleAntennaElement(...
    'FrequencyRange',[200e6 500e6]);
```

```
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[200e6 500e6],...
    'AxisDirection','Z');
elemindices = ones(4,4);
elemindices(2:3,2:3) = 2;
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',elemindices);
fc = 400e6;
c = physconst('LightSpeed');
patternElevation(sArray,fc,[0 75],...
    'PropagationSpeed',c,...
    'Type','directivity')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth`

plotResponse

System object: `phased.HeterogeneousURA`
Package: `phased`

Plot response pattern of array

Syntax

```
plotResponse(H,FREQ,V)
plotResponse(H,FREQ,V,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ,V)` plots the array response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`. The propagation speed is specified in `V`.

`plotResponse(H,FREQ,V,Name,Value)` plots the array response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Array object

FREQ

Operating frequency in Hertz specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

CutAngle

Cut angle as a scalar. This argument is applicable only when RespCut is 'Az' or 'El'. If RespCut is 'Az', CutAngle must be between –90 and 90. If RespCut is 'El', CutAngle must be between –180 and 180.

Default: 0

Format

Format of the plot, using one of 'Line', 'Polar', or 'UV'. If you set Format to 'UV', FREQ must be a scalar.

Default: 'Line'

NormalizeResponse

Set this value to true to normalize the response pattern. Set this value to false to plot the response pattern without normalizing it. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: true

OverlayFreq

Set this value to true to overlay pattern cuts in a 2-D line plot. Set this value to false to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is false, FREQ must be a vector with at least two entries.

This parameter applies only when Format is not 'Polar' and RespCut is not '3D'.

Default: true

Polarization

Specify the polarization options for plotting the array response pattern. The allowable values are |'None' | 'Combined' | 'H' | 'V' | where

- 'None' specifies plotting a nonpolarized response pattern
- 'Combined' specifies plotting a combined polarization response pattern
- 'H' specifies plotting the horizontal polarization response pattern
- 'V' specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is 'None'. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: 'None'

RespCut

Cut of the response. Valid values depend on Format, as follows:

- If Format is 'Line' or 'Polar', the valid values of RespCut are 'Az', 'El', and '3D'. The default is 'Az'.

- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

Unit

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
db	power pattern in dB scale
mag	field pattern
pow	power pattern
dbi	directivity

Default: `'db'`

Weights

Weight values applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. The dimension *N* is the number of elements in the array. The interpretation of *M* depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

AzimuthAngles

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

ElevationAngles

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the

Format parameter is set to 'Line' or 'Polar'. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the RespCut parameter to '3D', you can set the ElevationAngles and AzimuthAngles parameters simultaneously.

Default: [-90:90]

UGrid

U coordinate values for plotting array response, specified as a row vector. The UGrid parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to 'U' or '3D'. The values of UGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the UGrid and VGrid parameters simultaneously.

Default: [-1:0.01:1]

VGrid

V coordinate values for plotting array response, specified as a row vector. The VGrid parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to '3D'. The values of VGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set VGrid and UGrid parameters simultaneously.

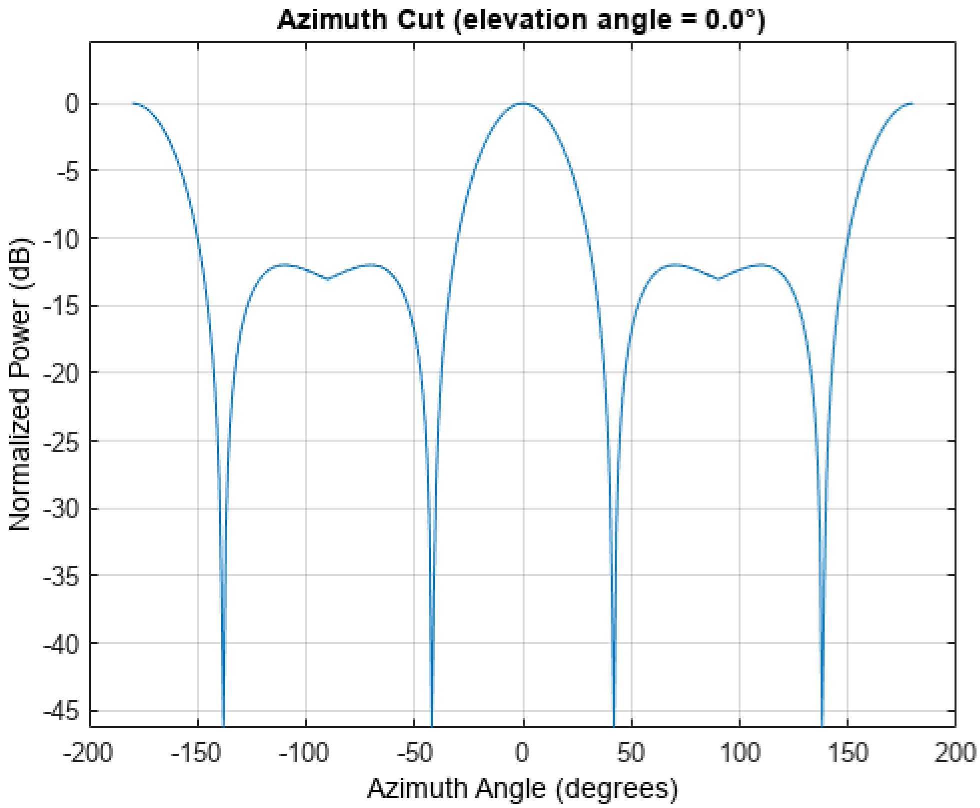
Default: [-1:0.01:1]

Examples

Azimuth Response and Directivity of Heterogeneous URA

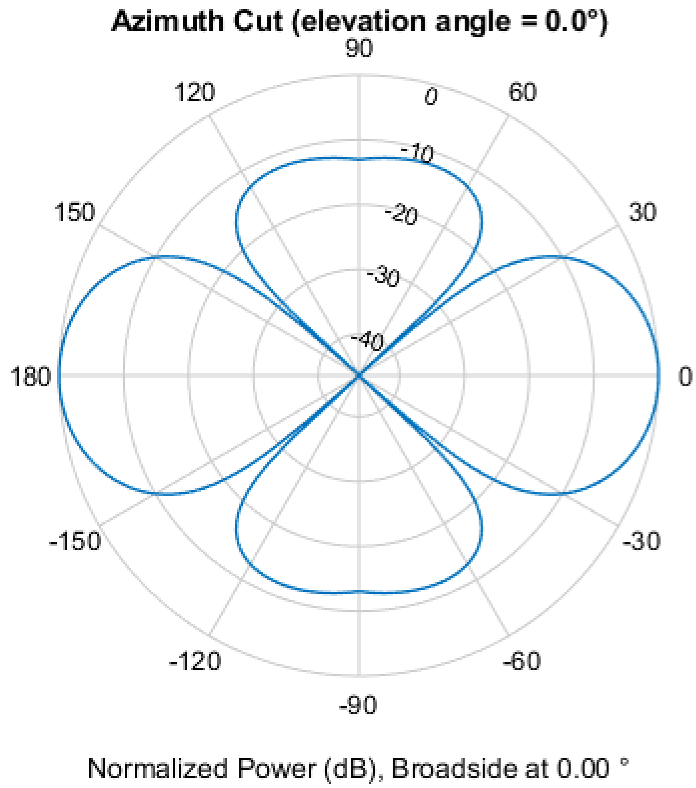
Construct a 3-by-3 heterogeneous URA with a rectangular lattice, then plot the array's azimuth response at 300 MHz.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1]);
fc = [3e8];
c = physconst('LightSpeed');
plotResponse(sArray,fc,c);
```



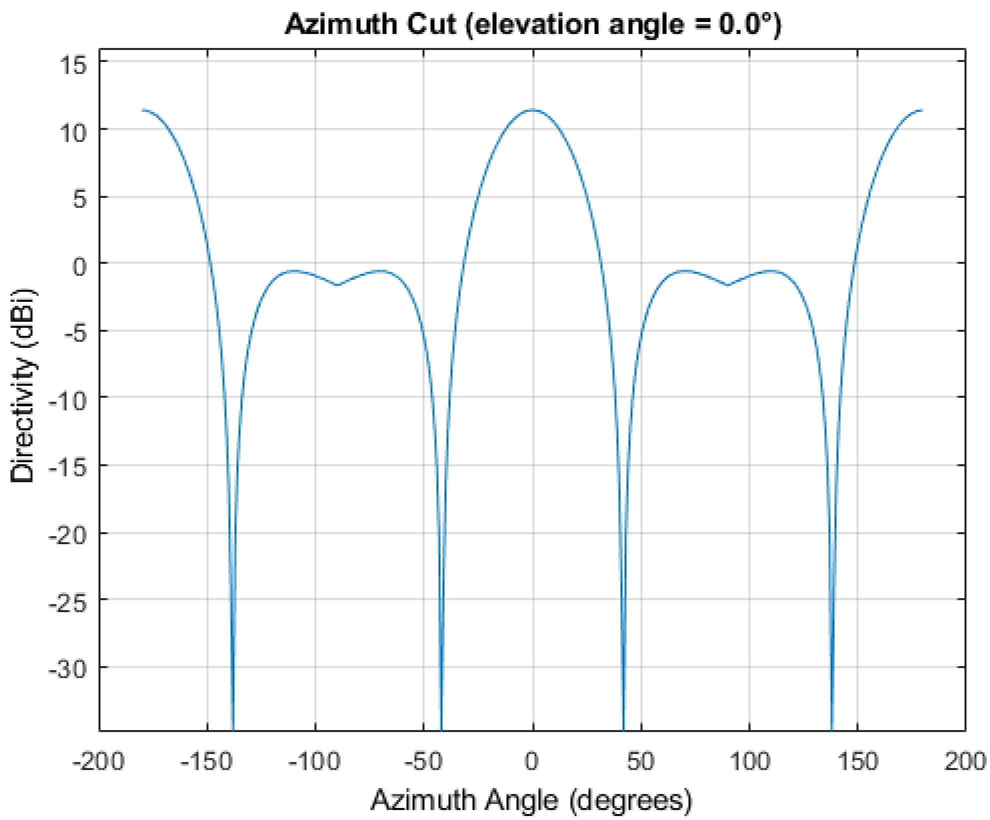
Plot the same result in polar form.

```
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar');
```



Finally, plot the directivity.

```
plotResponse(sArray,fc,c,'RespCut','Az','Unit','dbi');
```

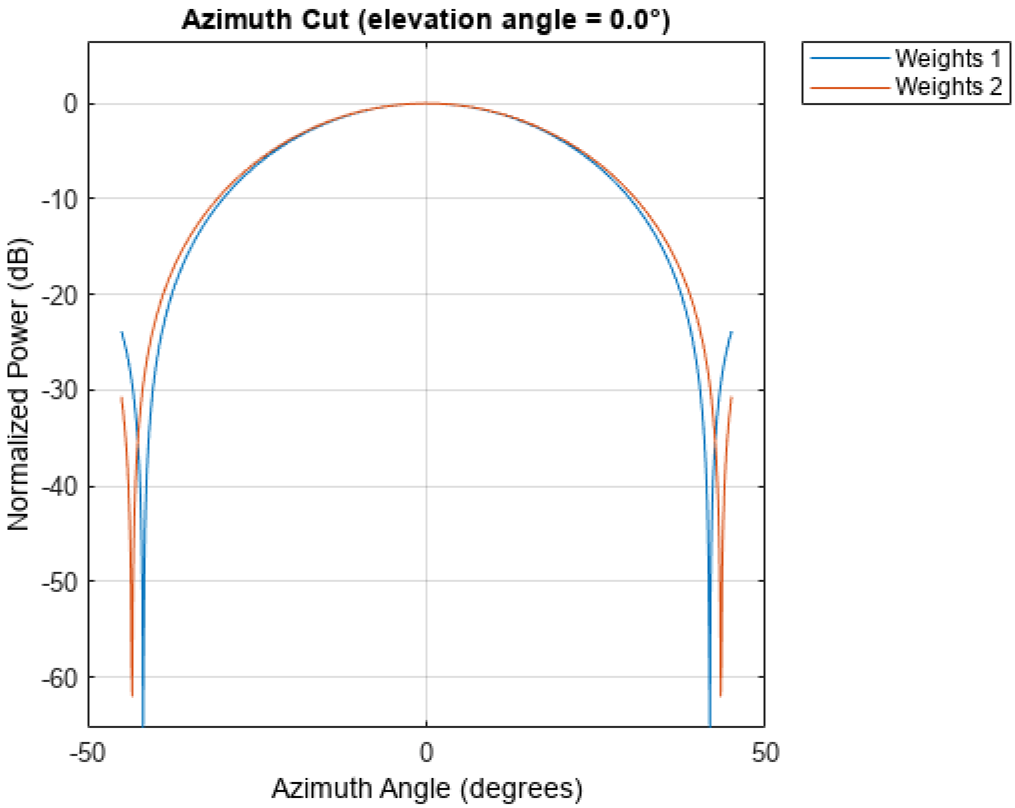


Azimuth Responses of a Heterogeneous URA For Two Sets of Weights

Construct a square 3-by-3 heterogeneous URA composed of 9 short-dipole antenna elements with different orientations. Using the AzimuthAngles parameter, plot the array's azimuth response in the -45 degrees to 45 degrees in 0.1 degree increments. The `Weights` parameter lets you display the array's response simultaneously for different sets of weights: in this case a uniform set of weights and a tapered set.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1]);
fc = [3e8];
c = physconst('LightSpeed');
wts1 = ones(9,1)/9;
wts2 = [.7,.7,.7,.7,1,.7,.7,.7,.7]';
wts2 = wts2/sum(wts2);
plotResponse(sArray,fc,c,'RespCut','Az',...
    'Format','Line',...
```

```
'AzimuthAngles',[-45:0.1:45],...
'Weights',[wts1,wts2],'Unit','db');
```



See Also

uv2azel | azel2uv

step

System object: `phased.HeterogeneousURA`
Package: `phased`

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the array elements' responses `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Array object

FREQ

Operating frequencies of array in hertz. `FREQ` is a row vector of length L. Typical values are within the range specified by a property of `H.Element`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has zero response at frequencies outside that range.

ANG

Directions in degrees. `ANG` is either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

If ANG is a row vector of length *M*, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be 0°.

Output Arguments

RESP

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, RESP, has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array. The dimension *M* is the number of angles specified in ANG. *L* is the number of frequencies specified in FREQ. For any element, the columns of RESP contain the responses of the array elements for the corresponding direction specified in ANG. Each of the *L* pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.
- If the array is capable of supporting polarization, the voltage response, RESP, is a MATLAB struct containing two fields, RESP.H and RESP.V. The field, RESP.H, represents the array's horizontal polarization response, while RESP.V represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array, and *M* is the number of angles specified in ANG. *L* is the number of frequencies specified in FREQ. Each column of RESP contains the responses of the array elements for the corresponding direction specified in ANG. Each of the *L* pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.

Examples

Response of 2-by-2 Heterogeneous URA of Cosine Antennas

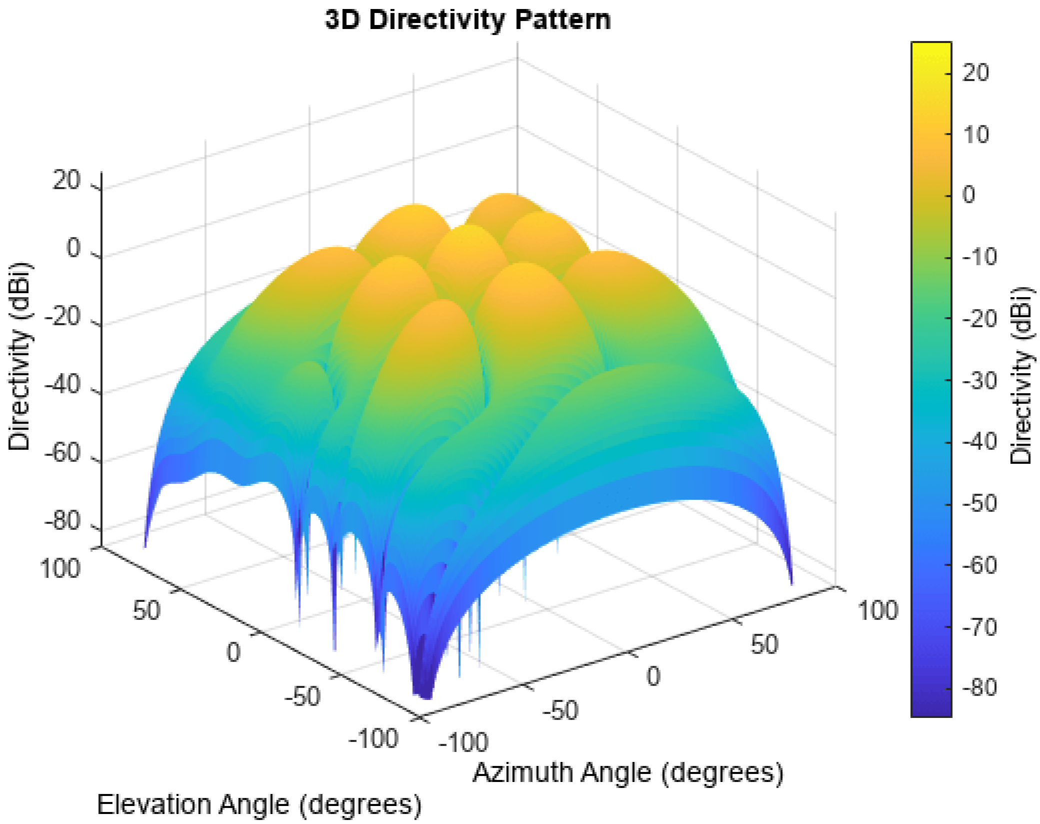
Construct a 2-by-2 rectangular lattice heterogeneous URA of cosine antenna elements. Find the response of each element at 30 degrees azimuth and 0 degrees elevation. Assume the operating frequency is 1 GHz. Then, plot the array directivity.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2; 2 1]);
fc = 1e9;
c = physconst('LightSpeed');
ang = [30;0];
resp = step(sArray,fc,ang)
```

```
resp = 4×1

    0.8059
    0.7719
    0.7719
    0.8059
```

Show the 3-D directivity pattern.

```
pattern(sArray,fc,[-180:180],[-90:90],...
    'PropagationSpeed',c,...
    'CoordinateSystem','rectangular',...
    'Type','directivity')
```



See Also
uv2azel | phitheta2azel

viewArray

System object: `phased.HeterogeneousURA`
Package: `phased`

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

`viewArray(H)` plots the geometry of the array specified in `H`.

`viewArray(H,Name,Value)` plots the geometry of the array, with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = viewArray( ___ )` returns the handle of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

`H`

Array object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Parent`

Handle to the axes along which the array geometry is displayed.

`ShowNormals`

Set this value to `true` to show the normal directions of all elements of the array. Set this value to `false` to plot the elements without showing normal directions.

Default: `false`

`ShowLocalCoordinates`

Logical flag specifying whether to show the local coordinate axes.

Default: `true`

`ShowAnnotation`

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Default: `true`

`Orientation`

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Default: `[0;0;0]`

`ShowTaper`

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

`ShowIndex`

Vector specifying the element indices to show in the figure. Each number in the vector must be an integer between 1 and the number of elements. You can also specify the value `'All'` to show the indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

`Title`

Character vector specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

`hPlot`

Handle of array elements in figure window.

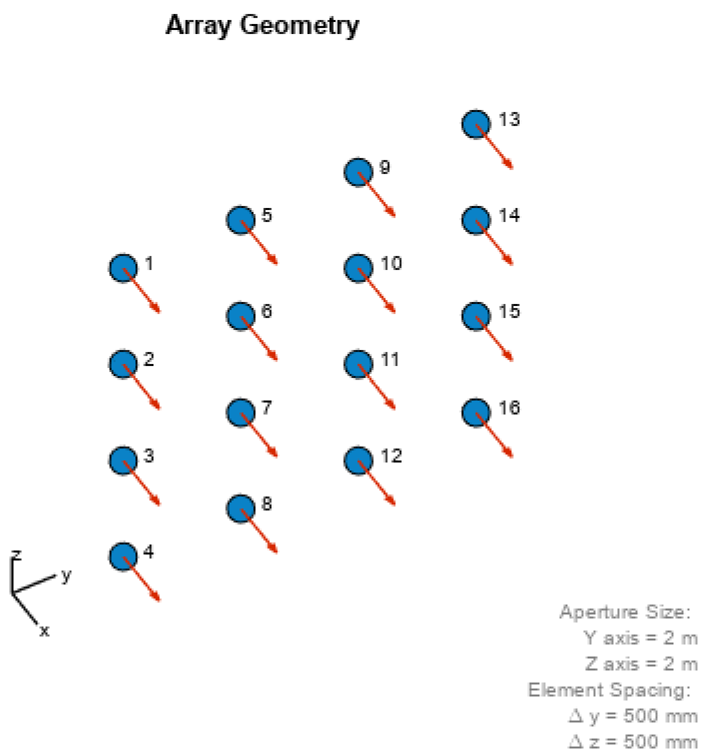
Examples

Geometry, Normal Directions, and Indices of Heterogeneous URA Elements

Display the element positions, normal directions, and indices for all elements of a 4-by-4 heterogeneous URA.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
```

```
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1; 1 2 2 1; 1 2 2 1; 1 1 1 1]);
viewArray(sArray,'ShowIndex','all','ShowNormal',true);
```



See Also

phased.ArrayResponse

Topics

Phased Array Gallery

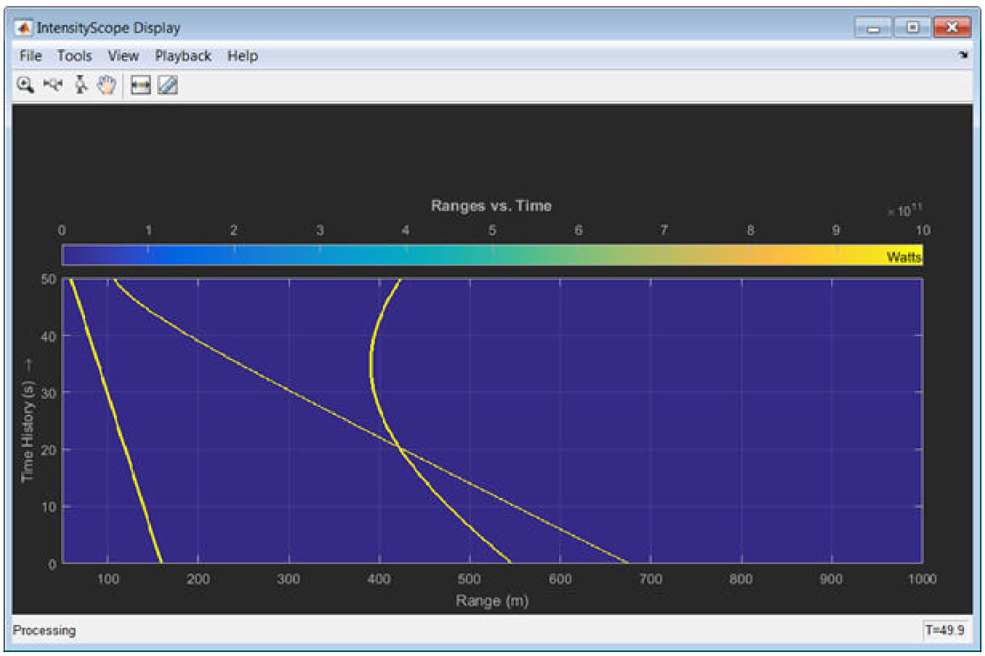
phased.IntensityScope

Package: `phased`

Range-time-intensity (RTI) or Doppler-time-intensity (DTI) display

Description

The `phased.IntensityScope` System object creates an intensity scope for viewing range-time-intensity (RTI) or Doppler-time-intensity (DTI) data. An intensity scope is a scrolling waterfall of intensity values as a function of time. Scan lines appear at the bottom of the display window and scroll off at the top. Each scan line represents signal intensity as a function of a parameter of interest, such as range or speed. You can also use this object to display angle-time-intensity data and spectral data. This figure shows an RTI display.



To create an intensity scope:

1. Define and set up the `phased.IntensityScope` System object. You can set any System object properties at this time or you can leave them at their default values. See "Construction" on page 1-674 .
2. Call the `step` method to add intensity lines to the bottom of the display according to the properties of the `phased.IntensityScope` System object. Some properties are tunable and can be changed at any time. Non-tunable properties cannot be changed after the first call to `step`. Subsequent calls to `step` add more intensity lines.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`sIS = phased.IntensityScope` creates an intensity scope System object, `sIS`, having default property values.

`sIS = phased.IntensityScope(Name,Value)` returns an intensity scope System object, `sIS`, with each specified property `Name` set to a specified `Value`. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Properties

Name — Window name
`'Intensity Scope'` (default) | character vector

Intensity scope window name, specified as a character vector. `Name` property and `Title` are different properties. The title appears inside the display window, above the data. The name appears in the title bar of the window.

Example: `'Range Intensity'`

Data Types: `char`

`XResolution` — *X*-axis sample spacing
`1` (default) | positive real-valued scalar

X-axis sample spacing, specified as a positive real-valued scalar. This quantity determines the width of each horizontal bin of the scan line. The units depend on the interpretation of the data. For example, if you are creating an RTI display, then setting `XResolution` to `0.5` is interpreted as 0.5 meters.

Example: `0.5`

Data Types: `double`

`XOffset` — *X*-axis offset
`0` (default) | real-valued scalar

X-axis offset, specified as a real-valued scalar. This quantity sets the value of the lowest bin of the scan line. The values of all other bins are equal to this value plus an integer multiple of `Xresolution`. The units depend upon the interpretation of the data. For example, if you are creating an RTI display, then setting `XOffset` to `100.0` is interpreted as 100 meters.

Example: `-0.1`

Data Types: `double`

`Xlabel` — X-axis label
`''` (default) | character vector

X-axis label, specified as a character vector.

Example: `'Range (km)'`

Data Types: `char`

`Title` — Title of display
`''` (default) | character vector

Title of the intensity scope display, specified as a character vector. `Title` property and `Name` are different properties. The title appears inside the display window, above the data. The name appears in the title bar of the window.

Example: `'Range vs Time'`

Data Types: `char`

`TimeResolution` — Time resolution
`.001` (default) | positive real-valued scalar

Time resolution of intensity line(s), specified as a positive real-valued scalar. Units are seconds.

Example: `.0001`

Data Types: `double`

`TimeSpan` — Time span of display window
0.1 (default) | positive real-valued scalar

Time span of intensity display, specified as a positive real-valued scalar. Units are seconds.

Example: `5.0`

Data Types: `double`

`IntensityUnits` — Intensity units label
`'dB'` (default) | character vector

Intensity units label displayed in the color bar, specified as a character vector.

Example: `'Watts'`

Data Types: `char`

`Position` — Location and size of intensity scope window
depends on display-resolution (default) | 1-by-4 vector of positive values

Location and size of the intensity scope window, specified as a 1-by-4 vector having the form `[left bottom width height]`. Units are in pixels.

- `left` and `bottom` specify the location of the bottom-left corner of the window.
- `width` and `height` specify the width and height of the window.

The default value of this property depends on the resolution of your display. This property is tunable.

Example: `[100 100 500 400]`

Data Types: `double`

Methods

hide	Hide intensity scope window
reset	Reset state of intensity scope System object
show	Show intensity scope window
step	Update intensity scope display

Common to All System Objects	
release	Allow System object property value changes

Examples

RTI Display of Moving Target

Use a `phased.IntensityScope` System object™ to display the echo intensity of a moving target as a function of range and time.

Run the simulation for 5 seconds at 0.1 second steps. In the display, each horizontal scan line shows the intensities of radar echo at each time step.

```
nsteps = 50;
dt = .1;
timespan = nsteps*dt;
```

Simulate a target at a range of 320.0 km and a range rate of 2.0 km/s. Echoes are resolved into range bins of 1 km resolution. The range bins span from 50 to 1000 km.

```
rngres = 1.0;
rngmin = 50.0;
rngmax = 1000.0;
tgtrange = 320.0;
rangerate = 2.0;
rngscan = [rngmin:rngres:rngmax];
```

Set up the Intensity Scope using these properties.

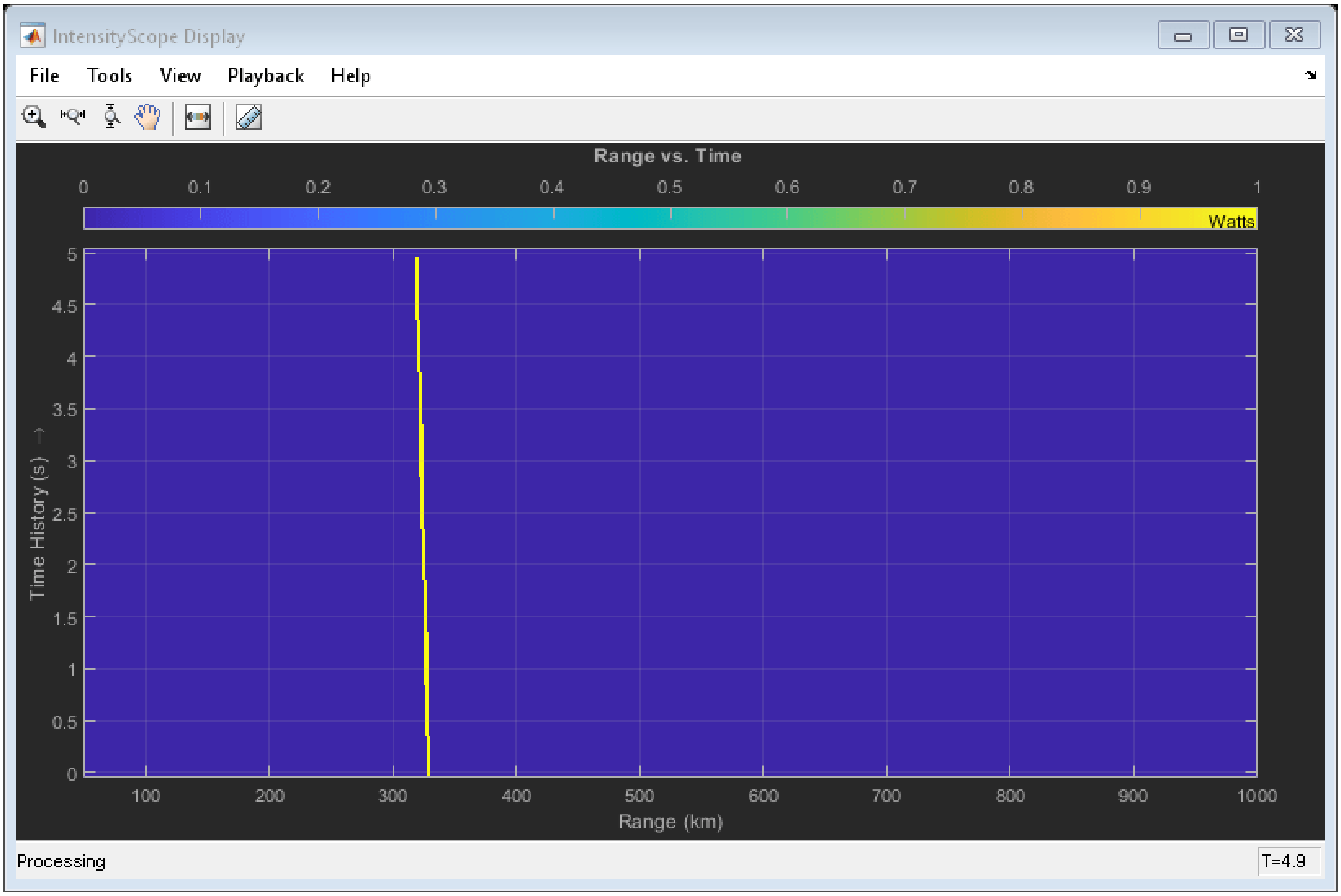
- Use the `XResolution` property to set the width of each scan line bin to the range resolution of 1 km.
- Use the `XOffset` property to set the value of the lowest range bin to the minimum range of 50 km.
- Use the `TimeResolution` property to set the value of the scan line time difference to 0.1 s.
- Use the `TimeSpan` property to set the height of the display window to the time duration of the simulation.
- Use the `IntensityUnits` property to set the display units to `Watts`.

```
scope = phased.IntensityScope('Name','IntensityScope Display',...
    'Title','Range vs. Time','XLabel','Range (km)',...
    'XResolution',rngres,'XOffset',rngmin,...
    'TimeResolution',dt,'TimeSpan',timespan, ...
    'IntensityUnits','Watts','Position',[100,100,800,450]);
```

Update the current target bin and create entries for two adjacent range bins. Each call to the `step` method creates a new scan line.

```
for k = 1:nsteps
    bin = floor((tgtrange - rngmin)/rngres) + 1;
    scanline = zeros(size(rngscan));
    scanline(bin+[-1:1]) = 1;
    scope(scanline.');
    tgtrange = tgtrange + dt*rangerate;
```

```
    pause(.1);
end
```



RTI Display of Three Moving Targets

Use the `phased.IntensityScope` System object™ to display the intensities of the echoes of three moving targets as functions of range and time.

Create the Radar and Target System Objects

Set up the initial positions and velocities of the three targets. Use the `phased.Platform` System object to model radar and target motions. The radar is stationary while the targets undergo constant velocity motion. The simulation runs for 500 steps at 0.1 second increments, giving a total simulation time of 50 seconds.

```
nsteps = 500;
dt = .1;
timespan = nsteps*dt;
x1 = [60,0,0]';
x2 = [60,-80,40]';
x3 = [300,0,-300]';
v1 = [2,0,0]';
v2 = [10,5,6]';
```

```
v3 = [-10,2,-4]';
platform = phased.Platform([0,0,0]',[0,0,0]');
targets = phased.Platform([x1,x2,x3],[v1,v2,v3]);
```

Set Up Range Bins

Each echo is put into a range bin. The range bin resolution is 1 meter and the range is from 50 to 1000 meters.

```
rngres = 1.0;
rngmin = 50.0;
rngmax = 1000.0;
rngscan = [rngmin:rngres:rngmax];
```

Create the Gain Function

Define a range-dependent gain function to enhance the display of targets at larger ranges. The gain function amplifies the returned echo for visualization purposes only.

```
rangegain = @(rng)(1e12*rng^4);
```

Create the Intensity Scope

Set up the Intensity Scope using these properties.

- Use the `XResolution` property to set the width of each scan line bin to the range resolution of 1 km.
- Use the `XOffset` property to set the value of the lowest range bin to the minimum range of 50 km.
- Use the `TimeResolution` property to set the value of the scan line time difference to 0.1 s.
- Use the `TimeSpan` property to set the height of the display window to the time duration of the simulation.
- Use the `IntensityUnits` property to set the display units to `Watts`.

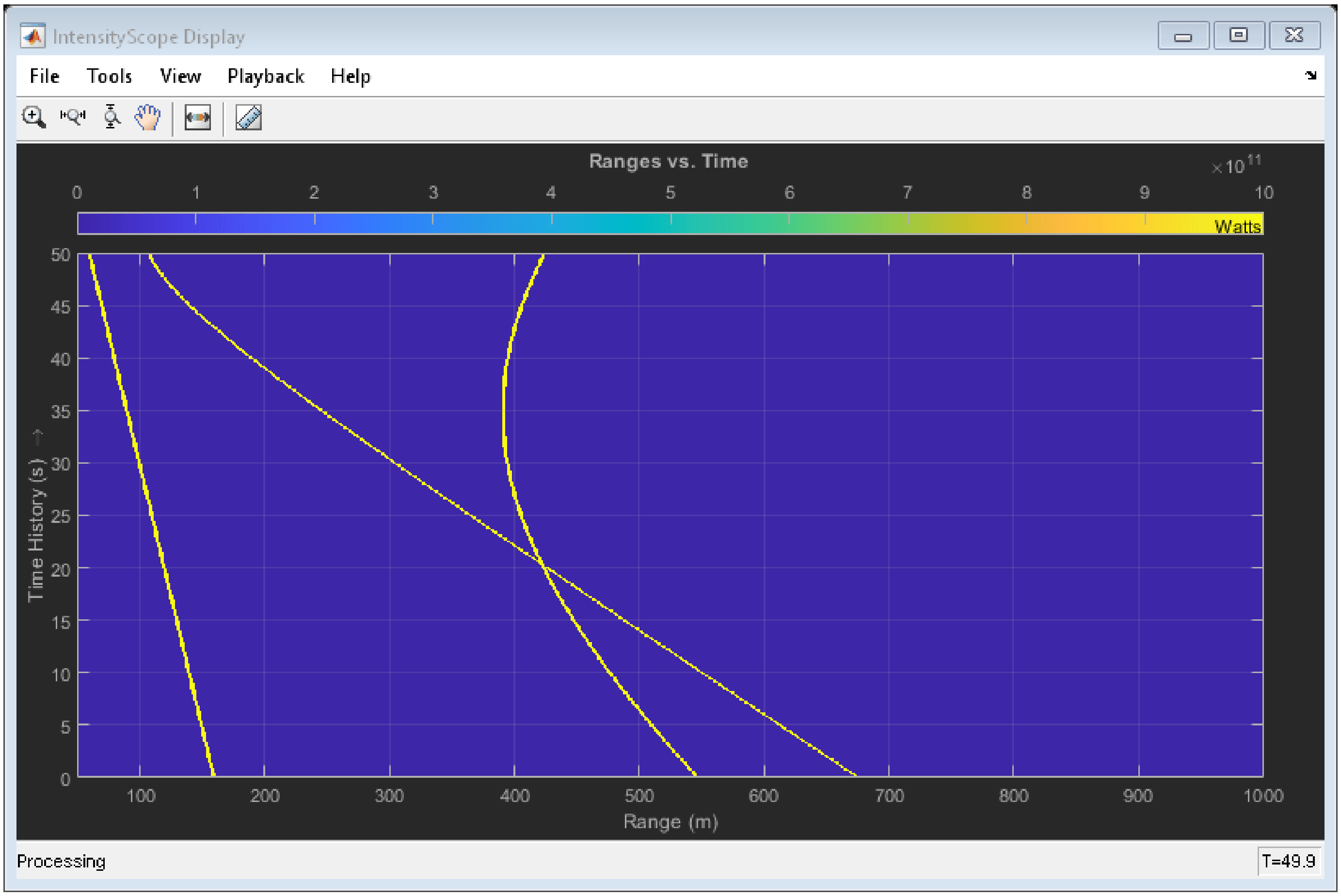
```
scope = phased.IntensityScope('Name','IntensityScope Display',...
    'Title','Ranges vs. Time','XLabel','Range (m)','XResolution',rngres,...
    'XOffset',rngmin,'TimeResolution',dt,'TimeSpan',timespan, ...
    'IntensityUnits','Watts','Position',[100,100,800,450]);
```

Run Simulation Loop

1. In this loop, move the targets at constant velocity using the `step` method of the `phased.Platform` System object.
2. Compute the target ranges using the `rangeangle` function.
3. Compute the target range bins by quantizing the range values in integer multiples of `rngres`.
4. Fill each target range bin and neighboring bins with a simulated radar intensity value.
5. Add the signal from each target to the scan line.
6. Call the `step` method of the `phased.IntensityScope` System object to display the scan lines.

```
for k = 1:nsteps
    xradar = platform(dt);
    xtgts = targets(dt);
    [rngs] = rangeangle(xtgts,xradar);
    scanline = zeros(size(rngscan));
```

```
    rngindx = ceil((rngs(1) - rngmin)/rngres);
    scanline(rngindx + [-1:1]) = rangegain(rngs(1))/(rngs(1)^4);

    rngindx = ceil((rngs(2) - rngmin)/rngres);
    scanline(rngindx + [-1:1]) = rangegain(rngs(2))/(rngs(2)^4);

    rngindx = ceil((rngs(3) - rngmin)/rngres);
    scanline(rngindx + [-1:1]) = rangegain(rngs(3))/(rngs(3)^4);

    scope(scanline.');
    pause(.1);
end
```



RTI and DTI Displays in Full Radar Simulation

Use the `phased.IntensityScope` System Object™ to display the detection output of a radar system simulation. The radar scenario contains a stationary single-element monostatic radar and three moving targets.

Set Radar Operating Parameters

Set the maximum range, peak power range resolution, operating frequency, transmitter gain, and target radar cross-section.

```
max_range = 5000;
range_res = 50;
fc = 10e9;
tx_gain = 20;
peak_power = 5500.0;
```

Choose the signal propagation speed to be the speed of light, and compute the signal wavelength corresponding to the operating frequency.

```
c = physconst('LightSpeed');
lambda = c/fc;
```

Compute the pulse bandwidth from the range resolution. Set the sampling rate, `fs`, to twice the pulse bandwidth. The noise bandwidth is also set to the pulse bandwidth. The radar integrates a number of pulses set by `num_pulse_int`. The duration of each pulse is the inverse of the pulse bandwidth.

```
pulse_bw = c/(2*range_res);
pulse_length = 1/pulse_bw;
fs = 2*pulse_bw;
noise_bw = pulse_bw;
num_pulse_int = 10;
```

Set the pulse repetition frequency to match the maximum range of the radar.

```
prf = c/(2*max_range);
```

Create System Objects for the Model

Choose a rectangular waveform.

```
waveform = phased.RectangularWaveform('PulseWidth',pulse_length,...
    'PRF',prf,'SampleRate',fs);
```

Set the receiver amplifier characteristics.

```
amplifier = phased.ReceiverPreamp('Gain',20,'NoiseFigure',0,...
    'SampleRate',fs,'EnableInputPort',true,'SeedSource','Property',...
    'Seed',2007);
transmitter = phased.Transmitter('Gain',tx_gain,'PeakPower',peak_power,...
    'InUseOutputPort',true);
```

Specify the radar antenna as a single isotropic antenna.

```
antenna = phased.IsotropicAntennaElement('FrequencyRange',[5e9 15e9]);
```

Set up a monostatic radar platform.

```
radarplatform = phased.Platform('InitialPosition',[0; 0; 0],...
    'Velocity',[0; 0; 0]);
```

Set up the three target platforms using a single System object.

```
targetplatforms = phased.Platform(...
    'InitialPosition',[2000.66 3532.63 3845.04; 0 0 0; 0 0 0], ...
    'Velocity',[150 -150 0; 0 0 0; 0 0 0]);
```

Create the radiator and collector System objects.

```
radiator = phased.Radiator('Sensor',antenna,'OperatingFrequency',fc);
collector = phased.Collector('Sensor',antenna,'OperatingFrequency',fc);
```

Set up the three target RCS properties.

```
targets = phased.RadarTarget('MeanRCS',[1.6 2.2 1.05],'OperatingFrequency',fc);
```

Create System object to model two-way freespace propagation.

```
channels= phased.FreeSpace('SampleRate',fs,'TwoWayPropagation',true,...
    'OperatingFrequency',fc);
```

Define a matched filter.

```
MFcoef = getMatchedFilter(waveform);
filter = phased.MatchedFilter('Coefficients',MFcoef,'GainOutputPort',true);
```

Create Range and Doppler Bins

Set up the fast-time grid. Fast time is the sampling time of the echoed pulse relative to the pulse transmission time. The range bins are the ranges corresponding to each bin of the fast time grid.

```
fast_time = unigrid(0,1/fs,1/prf,'[)');
range_bins = c*fast_time/2;
```

To compensate for range loss, create a time varying gain System Object.

```
gain = phased.TimeVaryingGain('RangeLoss',2*fspl(range_bins,lambda),...
    'ReferenceLoss',2*fspl(max_range,lambda));
```

Set up Doppler bins. Doppler bins are determined by the pulse repetition frequency. Create an FFT System object for Doppler processing.

```
DopplerFFTbins = 32;
DopplerRes = prf/DopplerFFTbins;
fft = dsp.FFT('FFTLengthSource','Property',...
    'FFTLength',DopplerFFTbins);
```

Create Data Cube

Set up a reduced data cube. Normally, a data cube has fast-time and slow-time dimensions and the number of sensors. Because the data cube has only one sensor, it is two-dimensional.

```
rx_pulses = zeros(numel(fast_time),num_pulse_int);
```

Create IntensityScope System Objects

Create two `IntensityScope` System objects, one for Doppler-time-intensity and the other for range-time-intensity.

```
dtiscope = phased.IntensityScope('Name','Doppler-Time Display',...
    'XLabel','Velocity (m/sec)', ...
    'XResolution',dop2speed(DopplerRes,c/fc)/2, ...
```

```
    'XOffset',dop2speed(-prf/2,c/fc)/2,...
    'TimeResolution',0.05,'TimeSpan',5,'IntensityUnits','Mag');
rtiscope = phased.IntensityScope('Name','Range-Time Display',...
    'XLabel','Range (m)', ...
    'XResolution',c/(2*fs), ...
    'TimeResolution',0.05,'TimeSpan',5,'IntensityUnits','Mag');
```

Run the Simulation Loop over Multiple Radar Transmissions

Transmit 2000 pulses. Coherently process groups of 10 pulses at a time.

For each pulse:

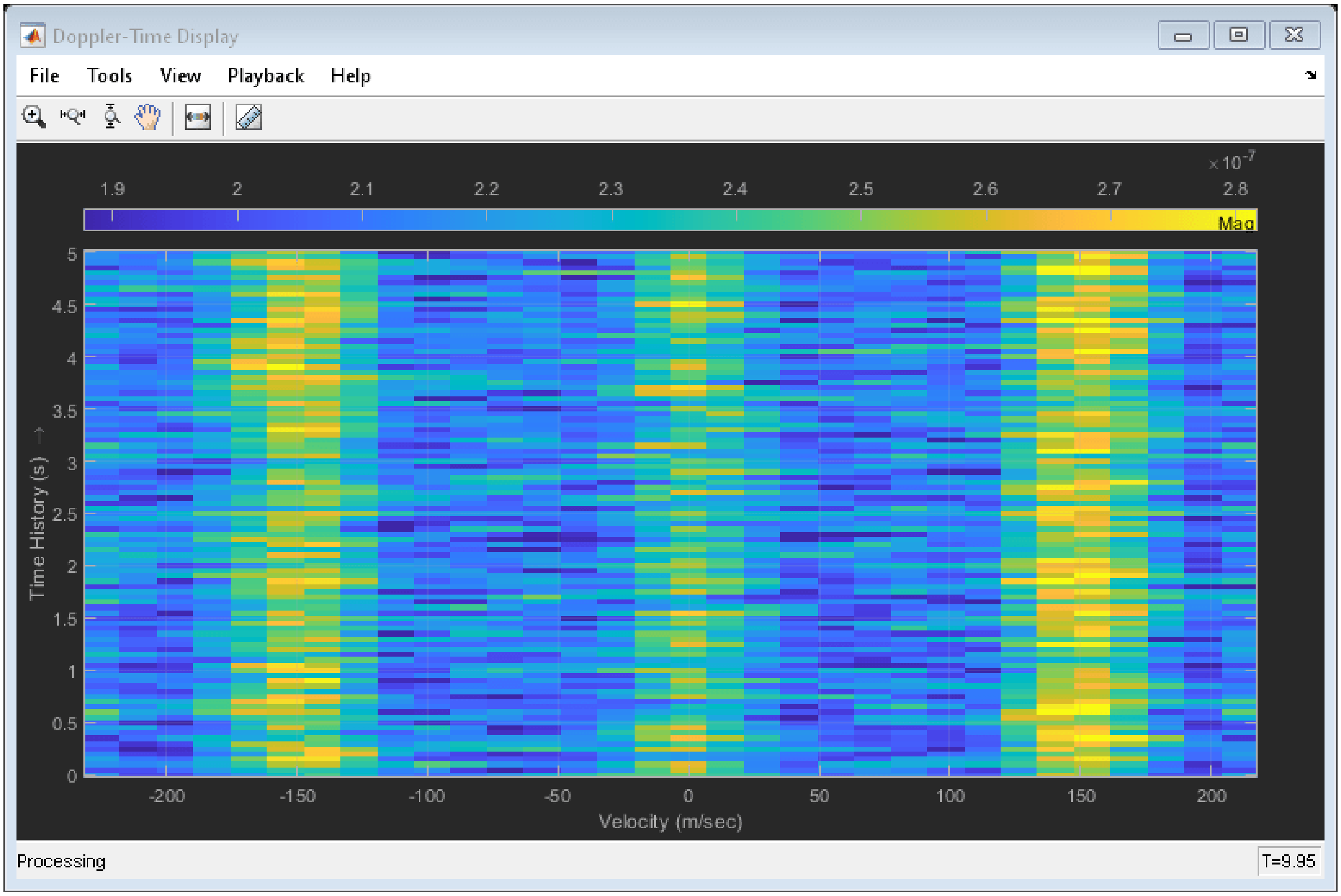
1. Update the radar position and velocity `radarplatform`
2. Update the target positions and velocities `targetplatforms`
3. Create the pulses of a single wave train to be transmitted `transmitter`
4. Compute the ranges and angles of the targets with respect to the radar
5. Radiate the signals to the targets `radiator`
6. Propagate the pulses to the target and back `channels`
7. Reflect the signals off the target `targets`
8. Receive the signal `sCollector`
9. Amplify the received signal `amplifier`
10. Form data cube

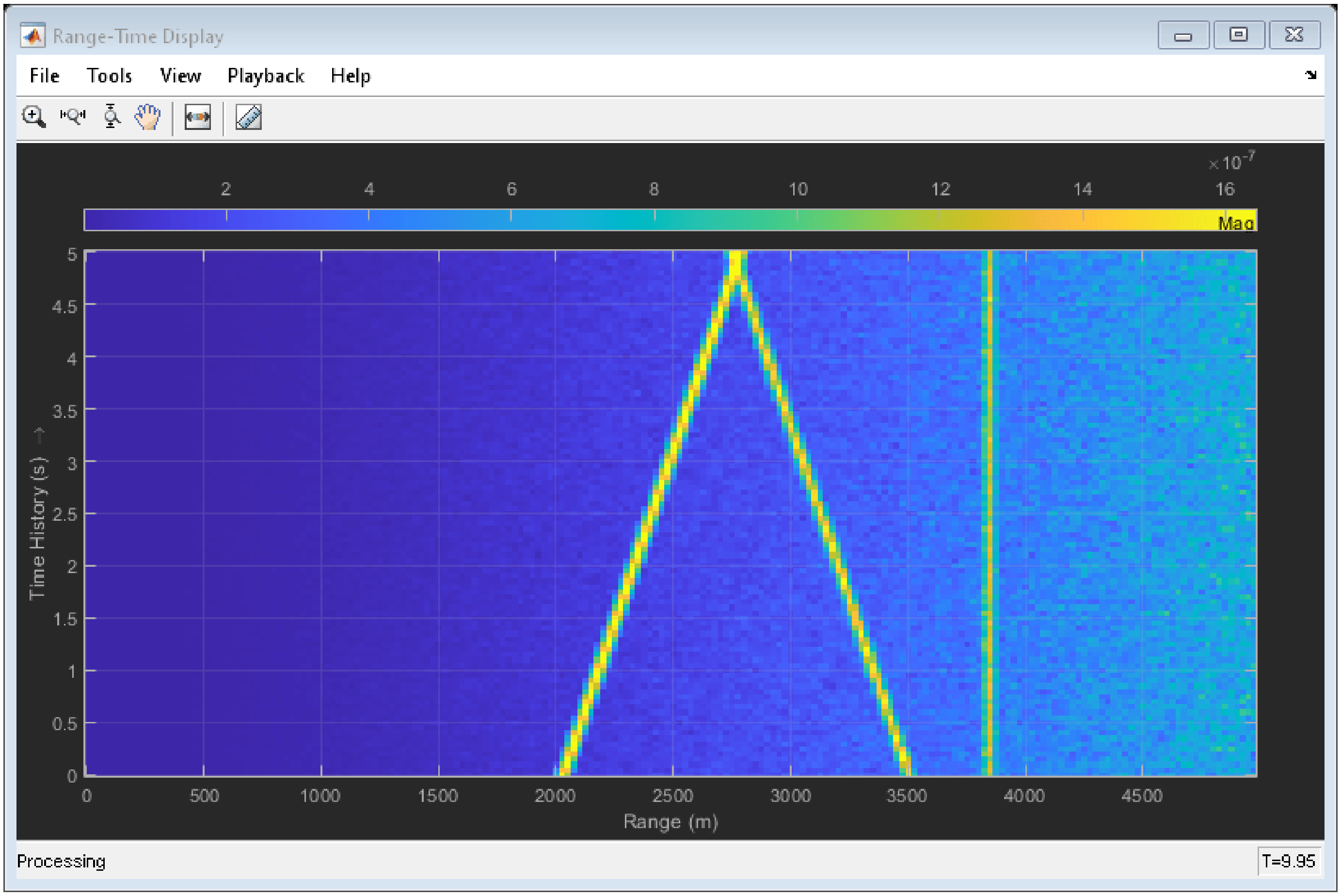
For each set of 10 pulses in the data cube:

1. Match filter each row (fast-time dimension) of the data cube.
2. Compute the Doppler shifts for each row (slow-time dimension) of the data cube.

```
pri = 1/prf;
nsteps = 200;
for k = 1:nsteps
    for m = 1:num_pulse_int
        [ant_pos,ant_vel] = radarplatform(pri);
        [tgt_pos,tgt_vel] = targetplatforms(pri);
        sig = waveform();
        [s,tx_status] = transmitter(sig);
        [~,tgt_ang] = rangeangle(tgt_pos,ant_pos);
        tsig = radiator(s,tgt_ang);
        tsig = channels(tsig,ant_pos,tgt_pos,ant_vel,tgt_vel);
        rsig = targets(tsig);
        rsig = collector(rsig,tgt_ang);
        rx_pulses(:,m) = amplifier(rsig,~(tx_status>0));
    end

    rx_pulses = filter(rx_pulses);
    MFdelay = size(MFcoef,1) - 1;
    rx_pulses = buffer(rx_pulses((MFdelay + 1):end), size(rx_pulses,1));
    rx_pulses = gain(rx_pulses);
    range = pulsint(rx_pulses,'noncoherent');
    rtiscope(range);
    dshift = fft(rx_pulses.');
    dshift = fftshift(abs(dshift),1);
```

```
        dtiscope(mean(dshift,2));
        radarplatform(.05);
        targetplatforms(.05);
    end
```





All of the targets lie on the x-axis. Two targets are moving along the x-axis and one is stationary. Because the radar is at the origin, you can read the target speed directly from the Doppler-Time Display window. The values agree with the specified velocities of -150, 150, and 0 m/sec.

Version History

Introduced in R2016a

See Also

`spectrogram`

Topics

"Measure Intensity Levels Using the Intensity Scope"

phased.ATIScope

View angle-time array response

Description

The `phased.ATIScope` System object creates an angle-time intensity (ATI) scope for displaying array response intensities as a function of time and angle of arrival. You can input two types of data – in-phase/quadrature (I/Q) data or intensity data. In either case, the scope displays signal magnitude or signal power.

- I/Q data – The data consists of fast-time I/Q samples of pulses and sweeps from multiple sensors. The scope creates intensity data and displays the array response map. To use I/Q data, set the `IQDataInput` property to `true`. In this mode, you can set the properties shown in "Properties Applicable for Processed I/Q Data" on page 1-694.
- Intensity data – The data consists of angle-time array intensity data. The scope displays the angle-time intensity response map. For example, you can obtain time-angle intensity data from the output of a beamformer. To use intensity data, set the `IQDataInput` property to `false`. In this mode, you can set the properties shown in "Properties Applicable for Viewing Response Data" on page 1-694.

To display angle-time response data:

1. Create the `phased.ATIScope` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
scope = phased.ATIScope()
scope = phased.ATIScope(Name=Value)
```

Description

`scope = phased.ATIScope()` creates an ATI System object for displaying the angle-time intensity response of an array.

`scope = phased.ATIScope(Name=Value)` creates an ATI `scope` System object with each specified property set to the specified value. See Properties for a list of properties. You can specify additional name-value pair arguments in any order as (`Name1=Value1`, … ,`NameN=ValueN`). For example,

```
scope = phased.ATIScope(IQDataInput=true,IntensityUnits="power", ...
        SensorArray=phased.URA(Size=[4,3]),OperatingFrequency=1e6)
```

creates an ATI scope System object that process I/Q data from a 4-by-3 uniform rectangular array. The intensity is displayed in power units. The operating frequency is 1 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

Name — Scope window name
`'Angle-Time Intensity Scope'` (default) | character vector | string

Scope window name, specified as a character vector or string. The name appears in the title bar of the window.

Example: `'Multi-target Angle-Time Intensity Scope'`

Tunable: Yes

Data Types: `char` | `string`

Position — Location and size of intensity scope window
dependent on display-resolution (default) | 1-by-4 vector of positive values

Location and size of the intensity scope window, specified as a 1-by-4 vector having the form `[left bottom width height]`.

- `left` and `bottom` specify the location of the bottom-left corner of the window.
- `width` and `height` specify the width and height of the window.

Units are in pixels.

The default value of this property depends on the resolution of your display. By default, the window is positioned in the center of the screen with a `width` of 800 pixels and a `height` of 450 pixels.

Example: `[100 100 500 400]`

Tunable: Yes

Data Types: `double`

IQDataInput — Input data type
`false` (default) | `true`

Input data type, specified as `false` or `true`. When `true`, the input consists of complex I/Q sample data and further processing is required to transform to the angle-time domain. When `false`, the input data is real response data that has already been transformed to the angle-time domain.

Data Types: `logical`

AngleLabel — Angle-axis label
`'Angles (deg)'` (default) | character vector | string

Angle-axis label, specified as a character vector or a string.

Example: `'Angles (rad)'`

Tunable: Yes

Data Types: `char` | `string`

`AngleResolution` — Angle difference between samples
`1.0` (default) | positive scalar

Angle separation between samples, specified as a positive scalar. This property defines the angle difference between columns of the input matrix. Units are in degrees.

Dependencies

To enable this property, set the `IQDataInput` property to `false`.

Data Types: `double`

`AngleOffset` — Angle offset
`0.0` (default) | scalar

Angle offset, specified as a scalar. This property defines the angle value of the first column of the input matrix. Units are in degrees.

Dependencies

To enable this property, set the `IQDataInput` property to `false`.

Data Types: `double`

`TimeResolution` — Time difference between rows
`0.001` (default) | positive scalar

Time interval between samples, specified as a positive scalar. This property defines the time interval between the rows of the scope. Units are in seconds.

Data Types: `double`

`TimeSpan` — Time span of display
`0.100` (default) | positive scalar

Time span of the intensity display, specified as a positive scalar. Units are in seconds.

Data Types: `double`

`IntensityUnits` — Response intensity units
`'db'` (default) | `'magnitude'` | `'power'`

Response intensity units, specified as `'db'`, `'magnitude'`, or `'power'`.

Data Types: `char` | `string`

`Title` — Display title
`'Angle vs. Time'` (default) | character vector | string

Display title, specified as a character vector or string.

Tunable: Yes

Data Types: `char` | `string`

`SensorArray` — Sensor array
`phased.ULA` array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object. See `phased.ULA` for the default values of a uniform linear array.

Example: `phased.URA`

Dependencies

To enable this property, set the `IQDataInput` to `true`.

`DOAMethod` — Direction of arrival estimation method
`'beamscan'` (default) | `'mvdr'` | `'music'`

Direction of arrival estimation method, specified as `'beamscan'`, `'mvdr'`, or `'music'`.

Example: `'mvdr'`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `char` | `string`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value of this property is the speed of light. See `physconst`. Units are in meters/second.

Example: 3e8

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency of the system, specified as a positive scalar. Units are in Hz.

Example: 2e9

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`NumPhaseShifterBits` — Number of phase shifter quantization bits
`0` (default) | non-negative scalar

The number of bits used to quantize the phase shift of the applied beamformer, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Example: 5

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`ForwardBackwardAveraging` — Forward-backward averaging
`false` (default) | `true`

Set this property to `true` to allow forward-backward averaging to estimate the covariance matrix for sensor arrays having a conjugate-symmetric array manifold.

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `logical`

`AzimuthScanAngles` — Azimuth scan angles
`-90:90` (default) | scalar | real-valued row vector

Azimuth scan angles, specified as a scalar or real-valued row vector. The angles must lie between –180° and 180°, inclusive. Specify the angles in increasing order. At least one of the sets of angles defined in the `AzimuthScanAngles` and `ElevationScanAngles` properties must be a scalar. Units are in degrees.

Example: `-20:20`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`ElevationScanAngles` — Elevation scan angles
`0` (default) | scalar | real-valued row vector

Elevation scan angles, specified as a scalar or real-valued row vector. The angles must lie between –90° and 90°, inclusive. Specify the angles in increasing order. At least one of the sets of angles defined in the `AzimuthScanAngles` and `ElevationScanAngles` properties must be a scalar. Units are in degrees.

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

Usage

Syntax

`scope(X)`

Description

`scope(X)` displays an ATI scope of array response intensities derived from the data `X`. You can display magnitude or power values.

Input Arguments

X — Intensity or I/Q data
M-by-*N* real-valued matrix | *M*-by-*N* complex-valued matrix

- When the `IQDataInput` property is `false`, `X` represents intensity data, specified as an *M*-by-*N* real-valued matrix. Each column of the matrix represents an intensity vector from successive times. Each matrix row *M* is the number of intensity values in an angle intensity vector and *N* is the number of intensity vectors. *M* should be greater than 1 and *N* should be equal to or greater than 1. The time between the intensity vectors is specified in the `TimeResolution` property.
- When the `IQDataInput` property is `true`, `X` represents I/Q data, specified as an *M*-by-*N* complex-valued matrix. Each row contains the data samples for all sensors at a single time. Each column contains the I/Q data for one sensor for all times.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to phased.ATIScope

hide	Turn off visibility of scope
isVisible	Visibility of scopes
show	Turn on visibility of scopes

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Display I/Q Data on Angle-Time Intensity Scope

Display the directions-of-arrival (DOA) of two signals arriving at a 10-by-5 uniform rectangular array (URA).

First, create a `phased.URA` array System object™ containing `phased.IsotropicAntennaElement` System objects spaced 0.45 wavelengths apart.

```
fc = 200e6;
lambda = physconst('LightSpeed')/fc;
```

```
d = 0.45*lambda;
element = phased.IsotropicAntennaElement( ...
    FrequencyRange=[100e6,300e6]);
array = phased.URA(Element=element,Size=[10,5], ...
    ElementSpacing=[d,d]);
```

Then, create an ATIScope to process I/Q data.

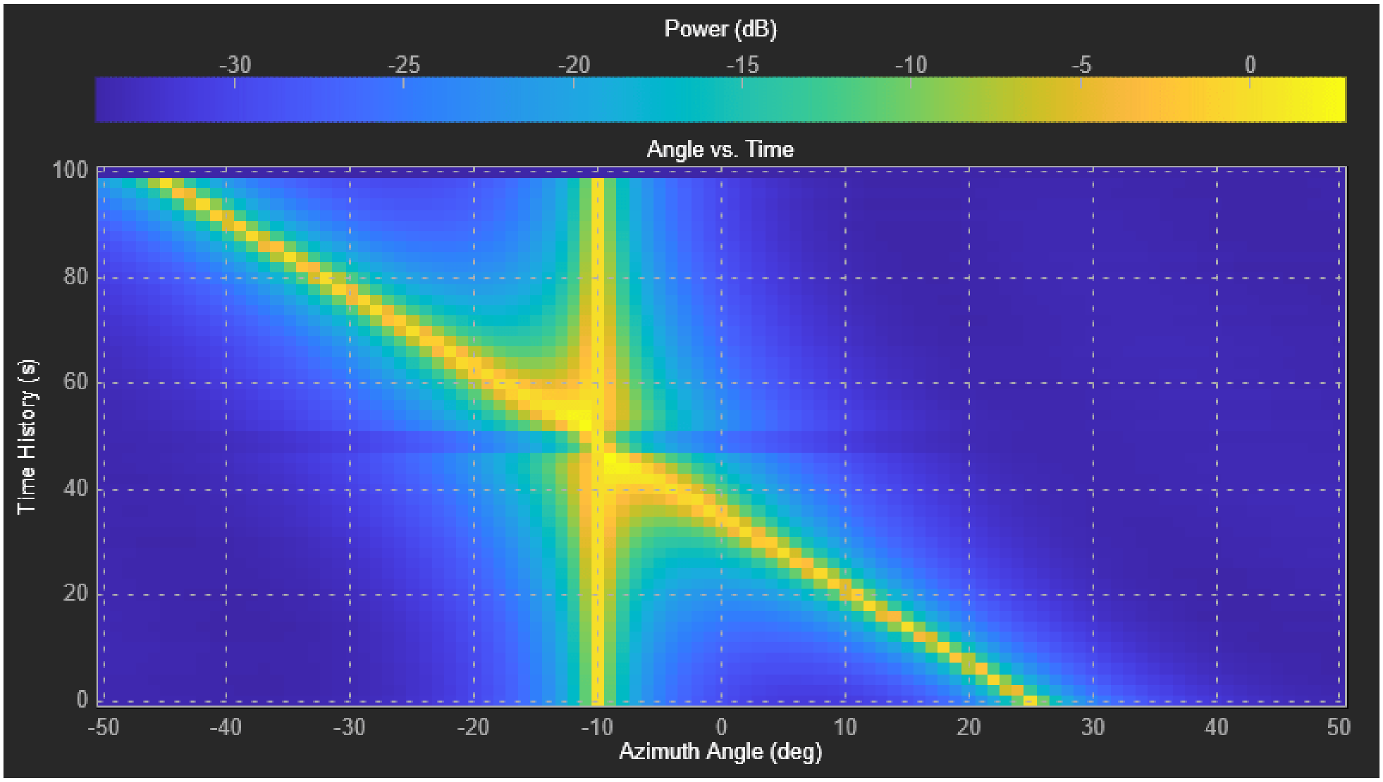
```
scope = phased.ATIScope(IQDataInput=true, ...
    SensorArray=array, ...
    DOAMethod="mvdr", ...
    OperatingFrequency=fc, ...
    AzimuthScanAngles=-50:50, ...
    ElevationScanAngles=0, ...
    AngleLabel="Azimuth Angle (deg)", ...
    TimeSpan=100, ...
    TimeResolution=2);
```

Generate baseband signals at 300 and 400 Hz. The sampling rate of the baseband signals is 8 kHz.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = exp(-1i*2*pi*t*300);
x2 = exp(-1i*2*pi*t*400);
```

Create two paths in azimuth-elevation space. The first path is a straight line starting at -45 degrees in azimuth and stopping at 25 degrees with 50 samples. The second path has constant azimuth of 30 degrees.

```
path1 = [linspace(-45,25,50); 0.0*ones(1,50)];
path2 = [-10*ones(1,50);zeros(1,50)];
xtot = [x1,x2];
for k = 1:size(path2,2)
    azel1 = [path1(:,k),path2(:,k)];
    x = collectPlaneWave(array,xtot,azel1,fc);
    noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
    scope(x + noise);
end
```



Display Array Response Using Angle-Time Intensity Scope

Display the directions-of-arrival (DOA) of two signals arriving at a 10-by-5 uniform rectangular array (URA).

First, create a `phased.URA` array System object™ composed of `phased.IsotropicAntennaElement` System objects.

```
fc = 200e6;
lambda = physconst('LightSpeed')/fc;
d = 0.45*lambda;
element = phased.IsotropicAntennaElement(FrequencyRange=[100e6,300e6]);
array = phased.URA(Element=element,Size=[5,5],ElementSpacing=[d,d]);
```

Then, create a `phased.ATIScope` System object.

```
scope = phased.ATIScope(IQDataInput=false, ...
    AngleLabel="Azimuth Angle (deg)", ...
    TimeSpan=100,TimeResolution=2, ...
    AngleOffset=-70);
```

Generate baseband signals at 300 and 400 Hz. The sampling rate of the baseband signal is 8 kHz.

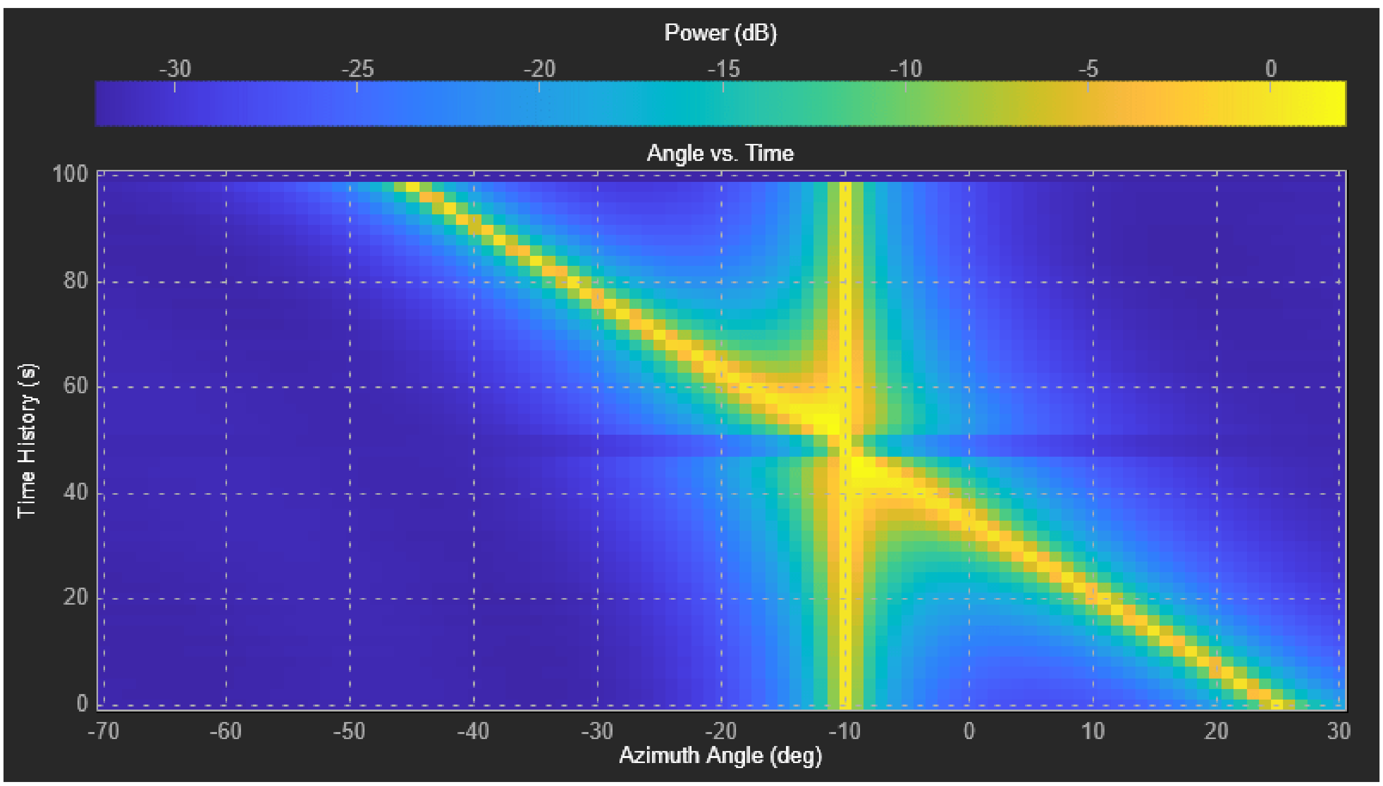
```
fs = 8000;
t = (0:1/fs:1).';
x1 = exp(-1i*2*pi*t*300);
x2 = exp(-1i*2*pi*t*400);
```

Create an MVDR beamformer System object. Specify the beamformer pointing angles.

```
azAngle = -50:50;
elAngle = zeros(size(azAngle));
bfAngle = [azAngle;elAngle];
beamformer = phased.MVDRBeamformer(SensorArray=array, ...
    PropagationSpeed=physconst('LightSpeed'), ...
    OperatingFrequency=fc,Direction=bfAngle, ...
    WeightsOutputPort=false);
```

Create two of objects in azimuth-elevation space. The first path is a straight line starting at -45° in azimuth and ending at 25°. The second path is at a constant azimuth of 10° degrees. Both paths have 50 samples.

```
path1 = [linspace(-45,25,50);zeros(1,50)];
path2 = [-10*ones(1,50);zeros(1,50)];
xtot = [x1,x2];
ymn = zeros(size(bfAngle,2));
for k = 1:size(path2,2)
    azel1 = [path1(:,k),path2(:,k)];
    x = collectPlaneWave(array,xtot,azel1,fc);
    noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
    y = beamformer(conj(x) + noise);
    ymn(k,:) = mean(abs(y),1);
    scope(ymn)
end
scope(ymn)
```



More About

Properties Applicable for Processed I/Q Data

These properties are applicable when `IQDataInput` is `true`.

Properties	
Name	Position
AngleLabel	TimeResolution
TimeSpan	IntensityUnits
Title	PropagationSpeed
SensorArray	DOAMethod
OperatingFrequency	ForwardBackwardAveraging
NumPhaseShifterBits	AzimuthScanAngles
ElevationScanAngles	

Properties Applicable for Viewing Response Data

These properties are applicable when `IQDataInput` is `false`.

Properties	
Name	Position
AngleLabel	AngleResolution
AngleOffset	TimeResolution
TimeSpan	IntensityUnits
Title	

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.BeamscanEstimator` | `phased.BeamscanEstimator2D` | `phased.MVDREstimator2D` | `phased.MVDREstimator` | `phased.MUSICEstimator2D` | `phased.MUSICEstimator` | `phased.AngleDopplerScope` | `phased.DTIScope` | `phased.IntensityScope` | `phased.RangeAngleScope` | `phased.RangeDopplerScope` | `phased.RTIScope`

phased.RTIScope

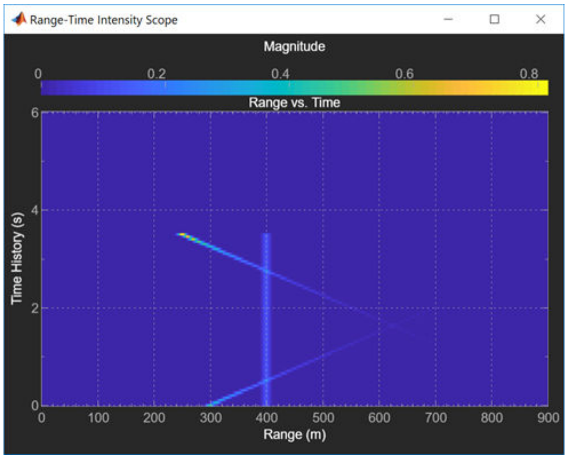
Package: phased

Range intensity scope

Description

The `phased.RTIScope` System object creates a scrolling display of range response intensity as a function of time. Each row represents the range response for a pulse or FMCW signal. Sequential calls to the object add new rows to the bottom of the display window. Columns represent the responses at a specific range over all pulses. You can input two types of data - in-phase and quadrature (I/Q) data or response data.

- I/Q data – The input consists of fast-time I/Q samples for one or more pulses or FM sweeps. The scope computes the range response and adds it to the display. To use I/Q data, set the `IQDataInput` property to `true`. In this mode, you can set the properties shown in "Properties Applicable to I/Q Data" on page 1-703.
- Response data – The data consists of the range response itself. The scope adds the range response to the display. For example, you can obtain the range response from a `phased.RangeResponse` object. To use response data, set the `IQDataInput` property to `false`. In this mode, you can set the properties shown in "Properties Applicable to Response Data" on page 1-703.



To create and run a range-time intensity scope,

1. Create the `phased.RTIScope` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
scope = phased.RTIScope
scope = phased.phased.RTIScope(Name,Value)
```

Description

`scope = phased.RTIScope` creates a range-time intensity scope System object, `scope`. This object displays the intensity of the range-time response for the input data.

`scope = phased.phased.RTIScope(Name,Value)` creates a range-time intensity scope, `scope`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose property names in quotes. For example,

```
scope = phased.RTIScope('IQDataInput',true,'RangeMethod', ...
        'FFT','SampleRate',1e6,'TimeResolution',0.5,'TimeSpan',10.0, ...
        'RangeFFTLength',1024);
```

creates a scope object that uses FFT-based range processing for I/Q data having a sample rate of 1 MHz. The time resolution is 0.5 seconds and the time span is 10 seconds. The range FFT length is 1024 samples.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Name` — Display caption
`'Range Time Intensity Scope'` (default) | character vector

Display caption, specified as a character vector. The caption appears in the title bar of the window.

Example: `'Multi-target Range Time Intensity Scope'`

Tunable: Yes

Data Types: `char`

`Position` — Location and size of intensity scope window
depends on display-resolution (default) | 1-by-4 vector of positive values

Location and size of the intensity scope window, specified as a 1-by-4 vector having the form `[left bottom width height]`.

- `left` and `bottom` specify the location of the bottom-left corner of the window.
- `width` and `height` specify the width and height of the window.

Units are in pixels.

The default value of this property depends on the resolution of your display. By default, the window is positioned in the center of the screen, with a width and height of 800 and 450 pixels, respectively.

Example: `[100 100 500 400]`

Tunable: Yes

Data Types: `double`

`IQDataInput` — Type of input data
`false` (default) | `true`

Type of input data, specified as `false` or `true`. When `true`, the object assumes that the input consists of I/Q sample data and further processing is required in the range domain. When `false`, the object assumes that the data is response data that has already been processed.

Data Types: `logical`

`RangeLabel` — Range axis label
`'Range (m)'` (default) | character vector

Range-axis label, specified as a character vector.

Example: `'Range (km)'`

Tunable: Yes

Data Types: `char`

`RangeResolution` — Range difference between samples
`1.0` (default) | positive scalar

Range distance between samples, specified as a positive scalar. This property defines the distance between columns of the scope. Units are in meters.

Data Types: `double`

`RangeOffset` — Range offset
`0.0` (default) | positive scalar

Range offset, specified as a positive scalar. This property defines the range value of the first column of the display. Units are in meters.

Data Types: `double`

`TimeResolution` — Time difference between rows
`0.001` (default) | positive scalar

Time interval between samples, specified as a positive scalar. This property defines the time interval between the rows of the scope. Units are in seconds.

Data Types: `double`

`TimeSpan` — Time span of display
`0.100` (default) | positive scalar

Time span of the intensity display, specified as a positive scalar. Units are in seconds.

Data Types: `double`

`IntensityUnits` — Response intensity units
`'db'` (default) | `'magnitude'` | `'power'`

Response intensity units, specified as a `'db'`, `'magnitude'`, or `'power'`.

Data Types: `char`

`RangeMethod` — Range processing method
`'Matched filter'` (default) | `'FFT'`

Range-processing method, specified as `'Matched filter'` or `'FFT'`.

`'Matched filter'`	The object applies a matched filter to the incoming signal. This approach is commonly used with pulsed signals, where the matched filter is a time-reversed replica of the transmitted signal.
`'FFT'`	Algorithm performs range processing by applying an FFT to the input signal. This approach is commonly used with FMCW continuous signals and linear FM pulsed signals.

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value of this property is the speed of light. See `physconst`. Units are in meters/second.

Example: `3e8`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`SampleRate` — Sample rate
`1e6` (default) | positive scalar

Sample rate, specified as a positive scalar. Units are in Hz.

Example: `10e3`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`SweepSlope` — FM sweep slope
`1e9` (default) | scalar

Slope of the linear FM sweep, specified as a scalar. Units are in Hz/sec.

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'FFT'`.

Data Types: `double`

`DechirpInput` — Dechirp input signal
`false` (default) | `true`

Set this property to `true` to dechirp the input signal before performing range processing. `false` indicates that the input signal is already dechirped and no dechirp operation is necessary.

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'FFT'`.

Data Types: `logical`

`RangeFFTLength` — FFT length used in range processing
`1024` (default) | positive integer

FFT length used for range processing, specified as a positive integer.

Example: `128`

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'FFT'`.

Data Types: `double`

`ReferenceRangeCentered` — Set reference range at center of range span
`true` (default) | `false`

Set this property to `true` to set the reference range to the center of the range span. Set this property to `false` to set the reference range to the beginning of the range span.

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'FFT'`.

Data Types: `logical`

Usage

Syntax

`scope(X)`

`scope(X,Xref)`

Description

`scope(X)` adds new rows to the range-time intensity scope. The input `X` can be I/Q sample data or range response data depending on the value of the `IQDataInput` property.

`scope(X,Xref)` also specifies a reference signal to use for dechirping the input signal, `X`. This syntax applies when you set the `IQDataInput` property to `true`, the `RangeMethod` property to `'FFT'`, and the `DechirpInput` property to `true`. This syntax is most commonly used with FMCW signals. `Xref` is generally the transmitted signal.

`scope(X,coeff)` also specifies matched filter coefficients, `coeff`. This syntax applies when you set the `IQDataInput` property to `true` and the `RangeMethod` property to `'Matched Filter'`. This syntax is most commonly used with pulsed signals.

Input Arguments

X — Input data
real-valued *N*-by-*M* matrix | complex-valued *N*-by-*M* matrix

Input data, specified as a complex-valued *N*-by-*M* matrix. The interpretation of the data depends on the setting of the `IQDataInput` property.

- When `IQDataInput` is `true`, each column contains *N* fast-time I/Q samples for a pulse or an FMCW sweep. *M* is the number of pulses in the case of pulsed signals or the number of dechirped frequency sweeps for FMCW signals. The scope computes and displays the range-response.
  - When `RangeMethod` is set to `'FFT'` and `DechirpInput` is `false`, `X` has previously been dechirped.
  - When `RangeMethod` is set to `'FFT'` and `DechirpInput` is `false`, `X` has not been previously dechirped. Use the syntax that includes `XREF` as input data.
  - When `RangeMethod` is set to `'MatchedFilter'`, `X` has not been matched filtered. Use the syntax that includes `COEF` as input data.
- When `IQDataInput` is `false`, each column contains *N* response samples for a pulse or an FMCW sweep such as that produced by the `phased.RangeResponse`. *M* is the number of pulses in the case of pulsed signals or the number of dechirped frequency sweeps for FMCW signals. The scope only displays the range-response.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Scope Objects

show	Turn on visibility of scopes
hide	Turn off visibility of scope
isVisible	Visibility of scopes

Common to All System Objects

step	Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Display Range Time Intensity Map for Three Targets

Create a scrolling display of intensity at each range as function of time. The intensity is a combination of intensities from three simulated targets. One target starts at a range of 250 m and moves outward to 950 m. The second target starts at 1000 m and moves inward to 300 m. The third stays at 400 m. The intensities are computed using the inline function `rangePow`. The targets move in steps of 10 m but the `rangePow` function spreads the intensity over nearby range bins which are spaced every meter.

The inline function `rangePow` simulates a spread target return having an intensity falling off with the fourth power of range.

```
txpow = 200;
gain = 2e8;
std = 5;
rangePow = @(rngbins,range) ...
    gain.*exp(-0.5*((rngbins-range)./std).^2).* ...
    txpow./(range.^4)./(sqrt(2*pi).*std);
```

Create an RTI Scope to view intensity data.

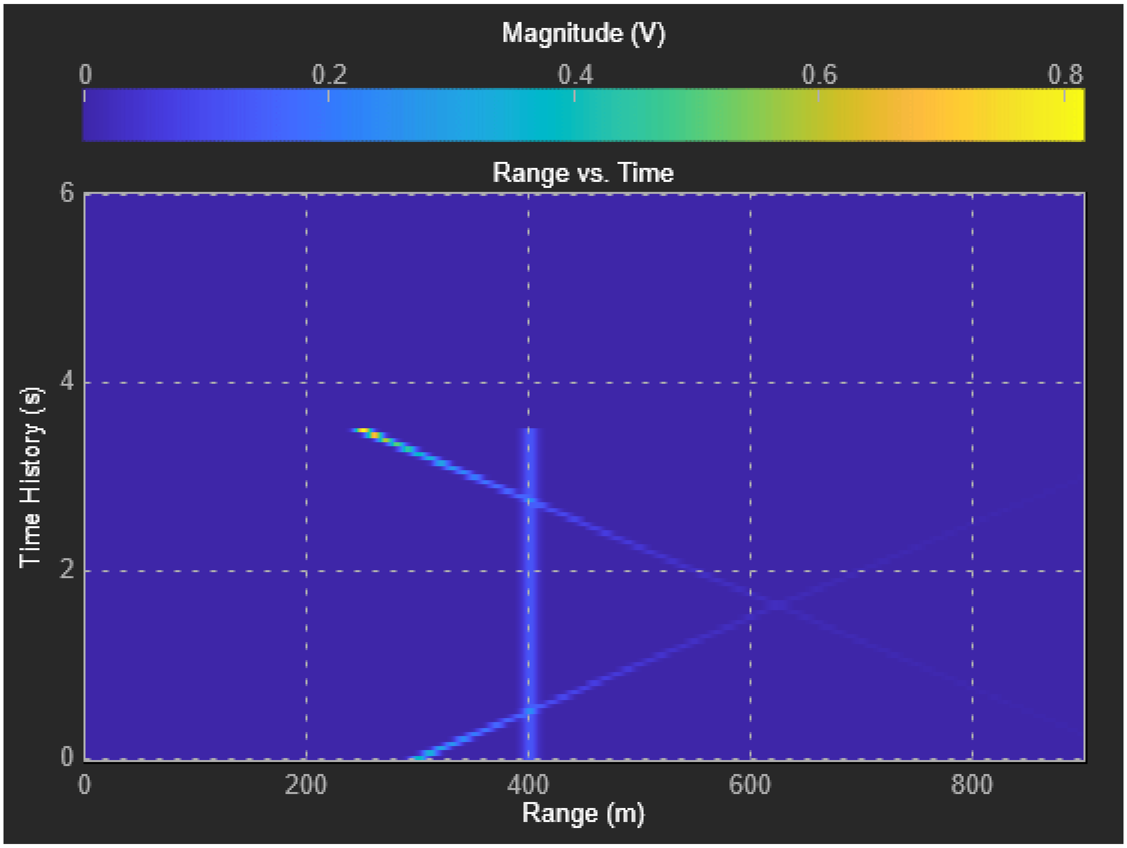
```
scope = phased.RTIScope( ...
    'IQDataInput',false,...
    'Name','Range-Time Intensity Scope',...
    'Position',[560 375 560 420],...
    'RangeLabel','Range (m)', ...
    'RangeResolution',1, ...
    'TimeResolution',0.05,'TimeSpan',6, ...
    'IntensityUnits','magnitude');
```

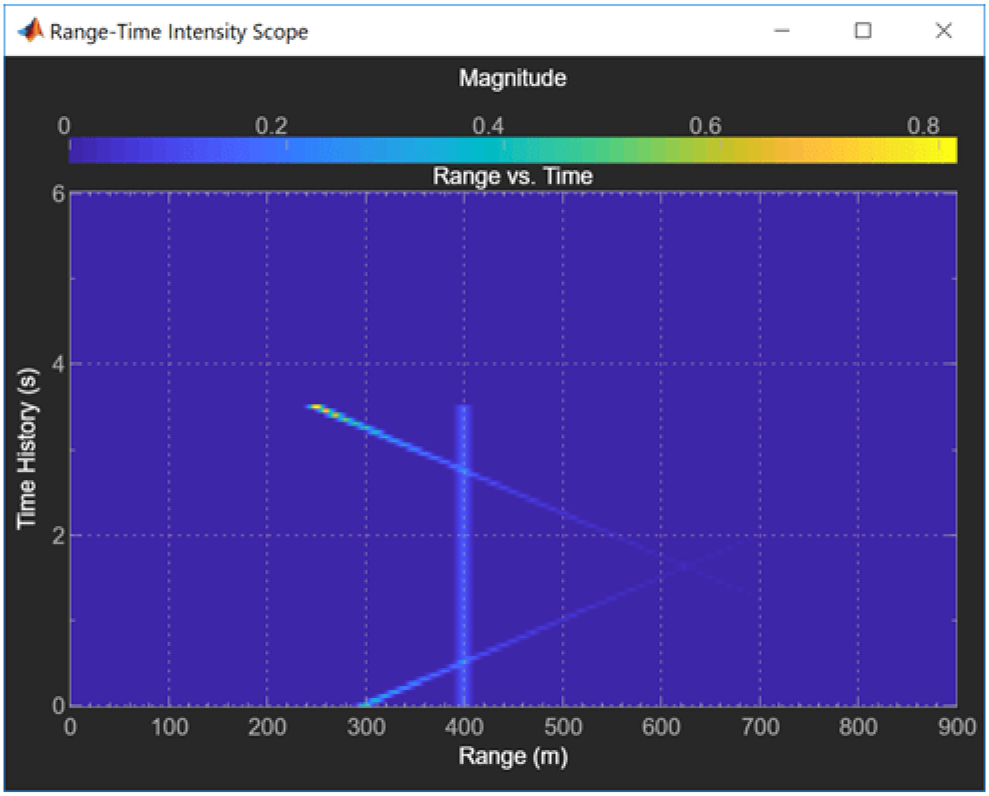
Create range bins for three targets.

```
rngbins = 0:900;
ranges(:,1) = 250:10:950;
ranges(:,2) = 1000:-10:300;
ranges(:,3) = 400;
```

Fill in all range bins by looping over all ranges and add each line at a time to the scope.

```
for k = 1:size(ranges,1)
    y = rangePow(rngbins,ranges(k,1)) + ...
        rangePow(rngbins,ranges(k,2)) + ...
        rangePow(rngbins,ranges(k,3));
    scope(y.');
    pause(.1);
end
```


Magnitude (V)
0
0.2
0.4
0.6
0.8
Range vs. Time
Time History (s)
Range (m)
0
2
4
6
0
200
400
600
800



More About

Properties Applicable to I/Q Data

These properties are applicable when `IQDataInput` is `true`.

Properties	
`Name`	`Position`
`RangeLabel`	`RangeResolution`
`RangeOffset`	`TimeResolution`
`TimeSpan`	`IntensityUnits`
`RangeMethod`	`PropagationSpeed`
`SampleRate`	`SweepSlope`
`DechirpInput`	`RangeFFTLength`
`ReferenceRangeCentered`	

Properties Applicable to Response Data

These properties are applicable when `IQDataInput` is `false`.

Properties	
Name	Position
RangeLabel	RangeResolution
RangeOffset	TimeResolution
TimeSpan	IntensityUnits

Version History

Introduced in R2019a

See Also

show | hide | isVisible | phased.RangeDopplerResponse | phased.RangeResponse | phased.DTIScope | phased.ATIScope | phased.AngleDopplerScope | phased.RangeAngleScope | phased.RangeDopplerScope

phased.DTIScope

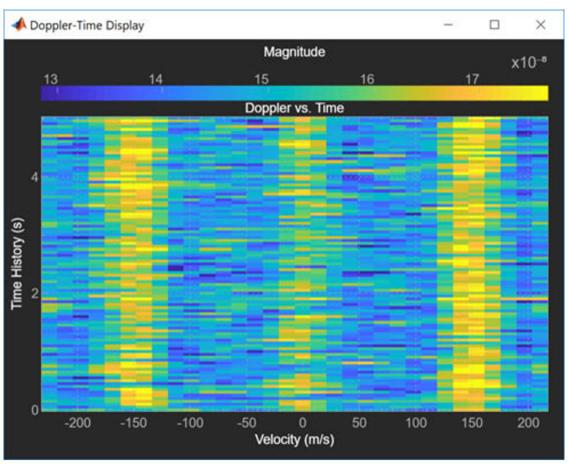
Package: phased

Doppler-time intensity scope

Description

The `phased.DTIScope` System object creates a scrolling display of Doppler response intensity as a function of time. Each row represents the Doppler response for a pulse or FMCW signal. Sequential calls to the object add new rows to the bottom of the display window. Columns represent the responses at specific Doppler values as a function of time. You can input two types of data - in-phase and quadrature (I/Q) data or response data.

- I/Q data – The input consists of fast-time I/Q samples from one or more pulses or FM sweeps. The scope computes the Doppler response and adds it to the display. To use I/Q data, set the `IQDataInput` property to `true`. In this mode, you can set the properties listed in "Properties Applicable to I/Q Data" on page 1-703.
- Response data – The data consists of the Doppler response itself as a function of time. The scope only adds the Doppler response to the display. For example, you can obtain Doppler responses from the `phased.RangeDopplerResponse` System object. To use response data, set the `IQDataInput` property to `false`. In this mode, you can set the properties listed in "Properties Applicable to Response Data" on page 1-703.



To create and run a Doppler-time intensity scope,

1. Create the `phased.DTIScope` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
scope = phased.DTIScope
scope = phased.phased.DTIScope(Name,Value)
```

Description

`scope = phased.DTIScope` creates a Doppler time intensity scope System object, `scope`. This object displays the Doppler-time response intensity of the input data.

`scope = phased.phased.DTIScope(Name,Value)` creates a Doppler-time intensity scope object, `scope`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose property names in quotes. For example,

```
scope = phased.DTIScope('IQInputData',false, ...
        'OperatingFrequency',1e6, ...
        'SampleRate',1e6,'DopplerOutput','Speed', ...
        'OperatingFrequency',10e6,'DopplerFFTLength',512);
```

creates a scope object that displays a 10-second time span of data using a Doppler FFT size of 512 for I/Q data. The Doppler output units are speed in meters per second.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

Name — Display caption

`Doppler Time Intensiy Scope'` (default) | character vector

Display caption, specified as a character vector. The caption appears in the title bar of the window.

Example: `'Multi-target Doppler Time Intensiy Scope'`

Tunable: Yes

Data Types: `char`

`Position` — Location and size of intensity scope window

depends on display-resolution (default) | 1-by-4 vector of positive values

Location and size of the intensity scope window, specified as a 1-by-4 vector having the form `[left bottom width height]`.

- `left` and `bottom` specify the location of the bottom-left corner of the window.
- `width` and `height` specify the width and height of the window.

Units are in pixels.

The default value of this property depends on the resolution of your display. By default, the window is positioned in the center of the screen, with a width and height of 800 and 450 pixels, respectively.

Example: `[100 100 500 400]`

Tunable: Yes

Data Types: `double`

`IQDataInput` — Type of input data
`false` (default) | `true`

Type of input data, specified as `true` or `false`. When `true`, the object assumes that the input consists of I/Q sample data and further processing is required in the Doppler domain. When `false`, the object assumes that the data is response data that has already been processed.

Data Types: `logical`

`DopplerResolution` — Doppler interval between samples
`1.0` (default) | positive scalar

Doppler interval between samples, specified as a positive scalar. This property defines the Doppler frequency difference between the scope columns. Units are in Hz.

Data Types: `double`

`DopplerOffset` — Doppler axis offset
`0.0` (default) | scalar

Doppler axis offset, specified as a scalar. This property apples a frequency offset to the Doppler axis. Units are in Hz.

Data Types: `double`

`TimeResolution` — Time difference between rows
`0.001` (default) | positive scalar

Time interval between samples, specified as a positive scalar. This property defines the time duration between rows of scope. Units are in seconds.

Data Types: `double`

`TimeSpan` — Time duration of display
`0.100` (default) | positive scalar

Time span of the intensity display, specified as a positive scalar. Units are in seconds.

Data Types: `double`

`IntensityUnits` — Response Intensity units
`'db'` (default) | `'magnitude'` | `'power'`

Response intensity units, specified as a `'db'`, `'magnitude'`, or `'power'`.

Data Types: `char`

DopplerOutput — Doppler output domain
'Frequency' (default) | 'Speed'

Doppler output domain, specified as 'Frequency' or 'Speed'. If you set this property to 'Frequency', the Doppler domain is the Doppler shift. Units are in Hz. If you set this property to 'Speed'', the Doppler domain is the corresponding radial speed. Units are in m/s.

Data Types: char

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value of this property is the speed of light. See physconst. Units are in meters/second.

Example: 3e8

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: double

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar.

Dependencies

To enable this property, set the IQDataInput property to true and the DopplerOutput to 'Speed'.

Data Types: double

DopplerFFTLength — FFT length used in Doppler processing
1024 (default) | positive integer

FFT length used in Doppler processing, specified as a positive integer.

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: double

Usage

Syntax

scope(X)

Description

scope(X) updates and displays the Doppler-time intensity scope for the input data, X. The input X can be I/Q sample data or Doppler response data depending on the value of the IQDataInput property.

Input Arguments

X — Input data
complex-valued *K*-by-*L* matrix

Input data, specified as a complex-valued *K*-by-*L* matrix. The interpretation of the data depends on the value of the `IQDataInput` property.

- When `IQDataInput` is `true`, the input consists of received fast-time and slow-time data for each PRI pulse or FMCW sweep. *K* denotes the number of time samples. *L* is the number of pulses in the case of pulsed signals or the number of dechirped frequency sweeps for FMCW signals. The scope computes and displays the Doppler response.
- When `IQDataInput` is `false`, the input already consists of response data in the Doppler domain such as that produced, for example, by `phased.RangeDopplerResponse`. Each row contains the set of Doppler responses. Each response corresponds to an element of the `Dop` vector. The scope serves only as a display of the Doppler response.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Scope Objects

show	Turn on visibility of scopes
hide	Turn off visibility of scope
isVisible	Visibility of scopes

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Display Doppler-Time Intensity Map for Three Targets

Create a `phased.DTIScope` object to view a scrolling Doppler-Time Intensity map.

Load the example data.

```
load('RTIDTIExampleData.mat')
rx_pulses = zeros(numel(fast_time),num_pulse_int);
```

Create the DTI scope.

```
scope = phased.DTIScope('IQDataInput',false,...
    'DopplerOutput','Speed',...
    'PropagationSpeed',c,...
```

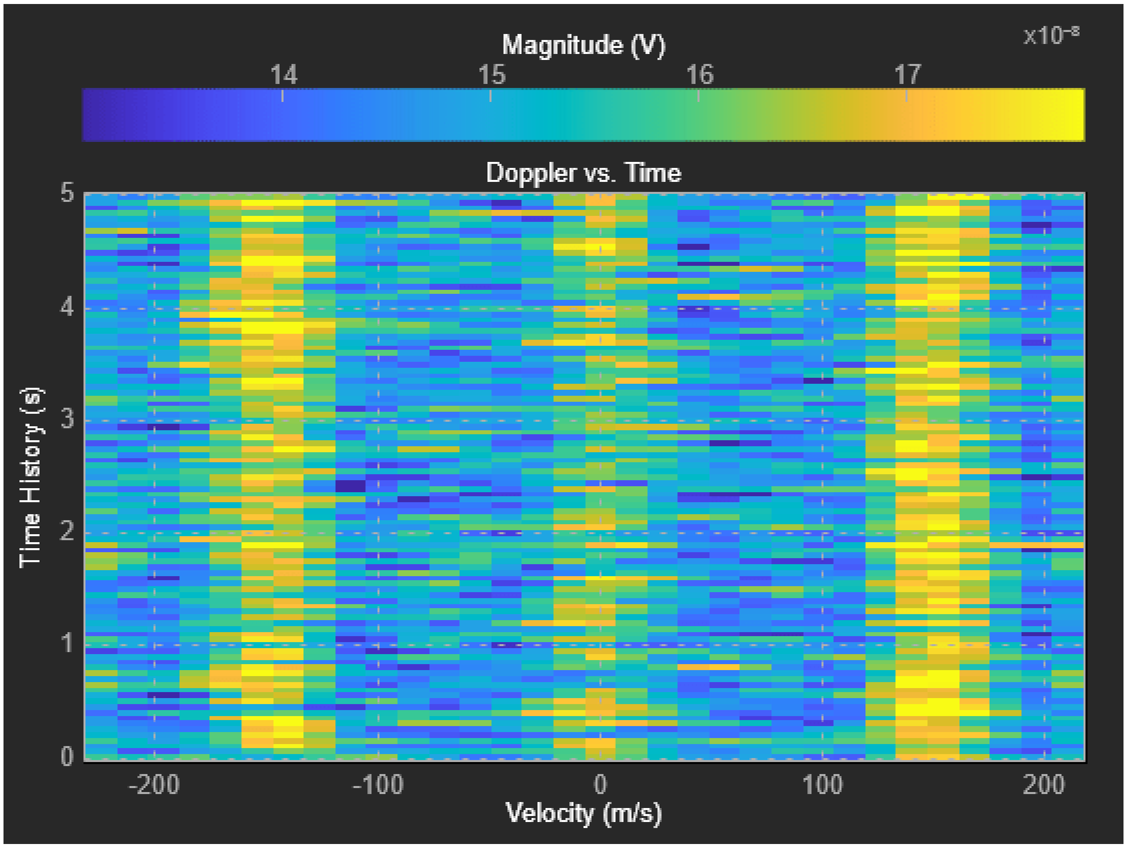
```
    'OperatingFrequency',fc,...
    'Name','Doppler-Time Display',...
    'DopplerResolution',DopplerRes, ...
    'DopplerOffset',-prf/2,...
    'TimeResolution',0.05,...
    'TimeSpan',5,...
    'IntensityUnits','magnitude',...
    'Position',[560 375 560 420]);
```

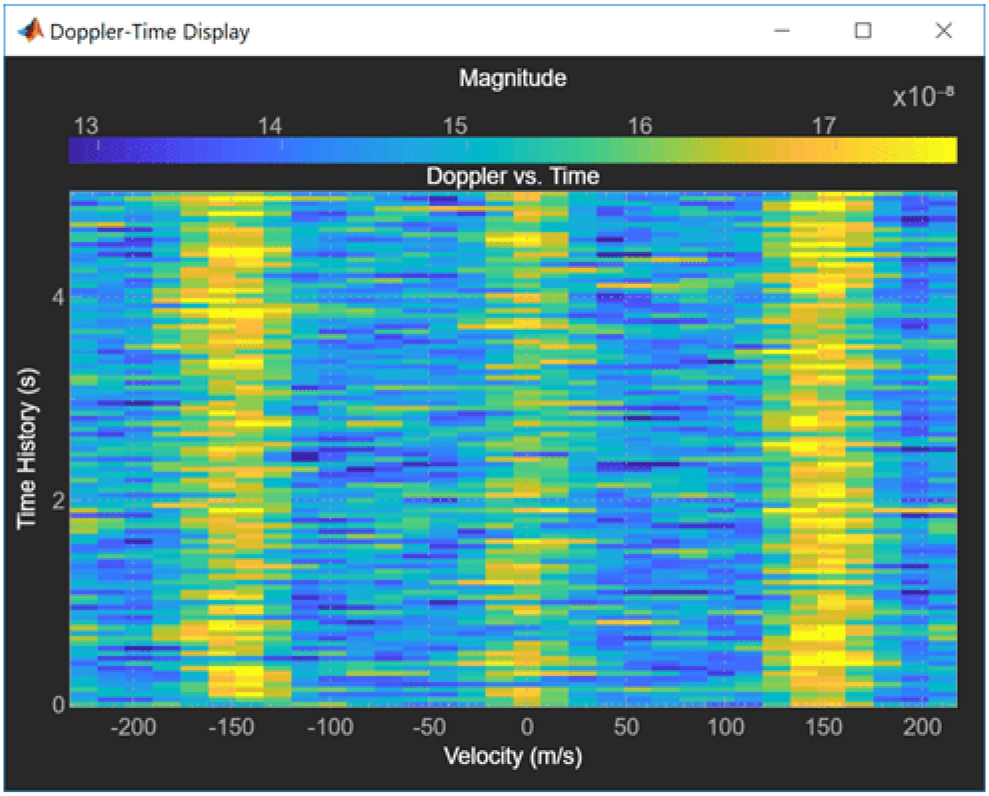
Obtain the pulse repetition interval, 33.3564 microsec.

```
pri = 1/prf;
```

Transmit 2000 pulses and coherently process a train of 10 pulses at a time. There are 200 trains. After each pulse, move the target and radar platform. The radar reflects off three targets. The first moves along the *x*-axis at -150 m/sec. The second moves along the *x*-axis at +150 m/sec. The third target is stationary. After each pulse train, compute the Doppler response using an FFT.

```
nsteps = 200;
for k = 1:nsteps
    for m = 1:num_pulse_int
        [ant_pos,ant_vel] = radarplatform(pri);
        [tgt_pos,tgt_vel] = targetplatforms(pri);
        sig = waveform();
        [s,tx_status] = transmitter(sig);
        [~,tgt_ang] = rangeangle(tgt_pos,ant_pos);
        tsig = radiator(s,tgt_ang);
        tsig = channels(tsig,ant_pos,tgt_pos,ant_vel,tgt_vel);
        rsig = targets(tsig);
        rsig = collector(rsig,tgt_ang);
        rx_pulses(:,m) = preamplifier(rsig,~(tx_status>0));
    end

    rx_pulses = gain(rx_pulses);
    dshift = fft(rx_pulses.');
    dshift = fftshift(abs(dshift),1);
    scope(mean(dshift,2));

    pause(0.1)
    radarplatform(.05);
    targetplatforms(.05);
end
```


Magnitude (V)
x10⁻⁸
14
15
16
17
Doppler vs. Time
Time History (s)
5
4
3
2
1
0
-200
-100
0
100
200
Velocity (m/s)



More About

Properties Applicable to I/Q Data

These properties are applicable when `IQDataInput` is `true`.

Properties	
`Name`	`Position`
`DopplerResolution`	`DopplerOffset`
`TimeResolution`	`TimeSpan`
`IntensityUnits`	`DopplerOutput`
`PropagationSpeed`	`OperatingFrequency`
`DopplerFFTLength`	

Properties Applicable to Response Data

These properties are applicable when `IQDataInput` is `false`.

Properties	
`Name`	`Position`
`DopplerResolution`	`DopplerOffset`

Properties	
TimeResolution	TimeSpan
IntensityUnits	DopplerOutput
PropagationSpeed	OperatingFrequency

Version History

Introduced in R2019a

See Also

show | hide | isVisible | phased.RangeDopplerResponse | phased.AngleDopplerScope | phased.RangeAngleScope | phased.RangeDopplerScope | phased.RTIScope | phased.ATIScope

hide

System object: phased.IntensityScope
Package: phased

Hide intensity scope window

Syntax

```
hide(sIS)
```

Description

`hide(sIS)` hides the display window of the `phased.IntensityScope` object, `sIS`.

Input Arguments

sIS — Intensity scope
`phased.IntensityScope` System object

Intensity scope, specified as a `phased.IntensityScope` System object.

Example: `phased.IntensityScope`

Examples

Hide and Show Intensity Scope

Create an angle-time-intensity scope. Use the `phased.IntensityScope` System object™ to display simulated intensity as a function of the angular motion of a moving target. After five steps in the processing loop, use the `hide` method to hide the scope. At completion of the loop, use the `show` method to show the scope.

Simulate data for 5 seconds with a time interval of 0.5 seconds between scan lines.

```
nsteps = 10;
dt = 0.5;
timespan = nsteps*dt;
```

Set Up IntensityScope System Object

Create an angle-time-intensity scope having azimuth angle bins spanning −180° to 180° with 1° resolution.

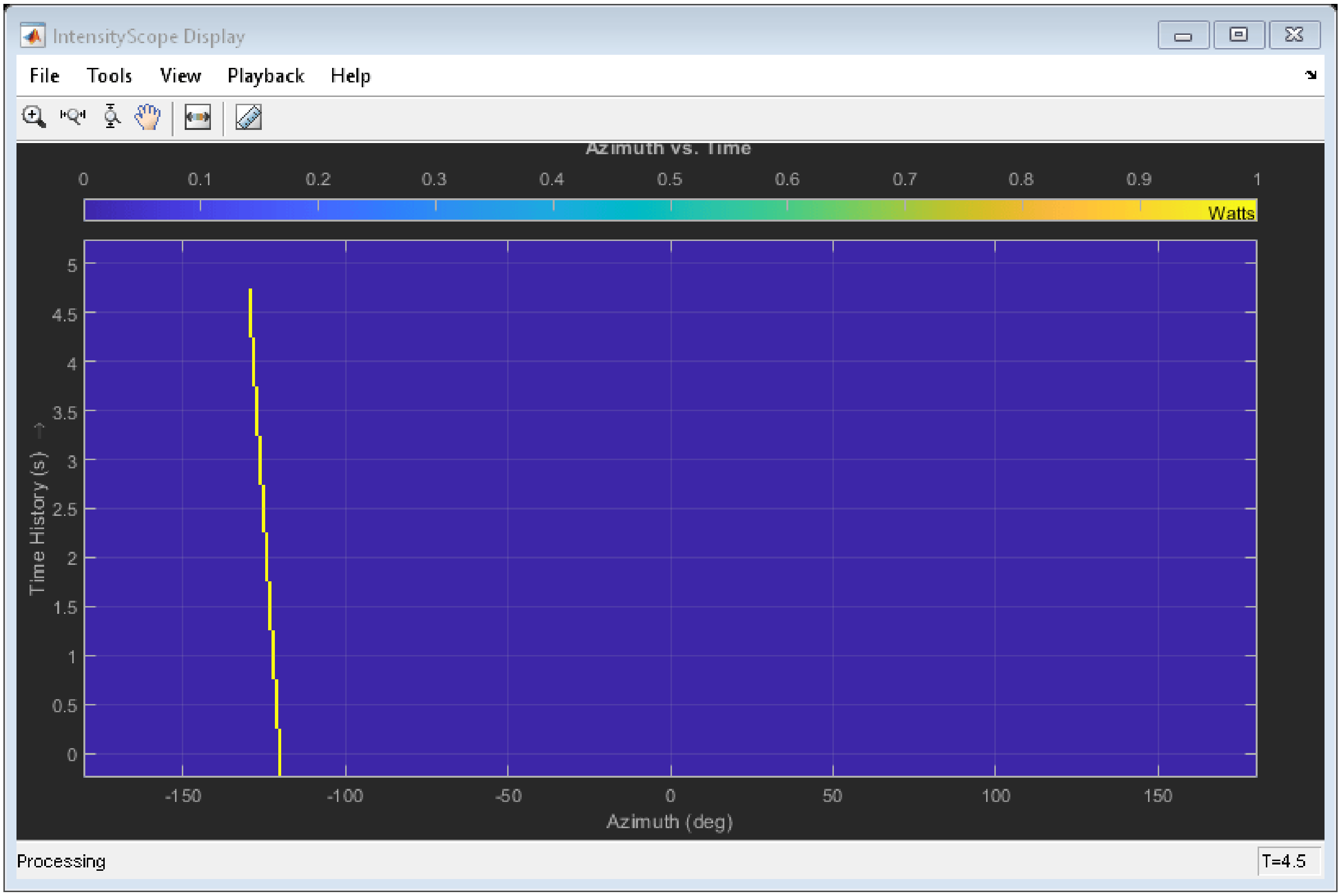
```
scanline = zeros(361,1);
angres = 1.0;
angmin = -180.0;
angmax = 180.0;
rtidisplay = phased.IntensityScope( ...
    'Name','IntensityScope Display',...
    'Title','Azimuth vs. Time',...
```

```
    'XLabel','Azimuth (deg)', ...
    'XResolution',angres,'XOffset',angmin,...
    'TimeResolution',dt,'TimeSpan',timespan, ...
    'IntensityUnits','Watts',...
    'Position',[100,100,800,450]);
```

Loop Over Scan Updates

Simulate angular motion and fill the bin containing the current angular position of the signal. Hide the scope after the 5th step and show the scope at the end of the simulation.

```
for k = 1:nsteps
    ang = -130.0 + k;
    binindexdx = floor((ang - angmin)/angres) + 1;
    scanline(binindexdx) = 1;
    rtidisplay(scanline);
    scanline(binindexdx) = 0;
    if k == 5
        hide(rtidisplay)
    end
    pause(.1);
end
show(rtidisplay)
```



Version History

Introduced in R2016a

reset

System object: `phased.IntensityScope`
Package: `phased`

Reset state of intensity scope System object

Syntax

`reset(sIS)`

Description

`reset(sIS)` resets the internal state of the `phased.IntensityScope` System object, `sIS`, to its initial value.

Input Arguments

`sIS` — Intensity scope
`phased.IntensityScope` System object

Intensity scope, specified as a `phased.IntensityScope` System object.

Example: `phased.IntensityScope`

Version History

Introduced in R2016a

show

System object: phased.IntensityScope
Package: phased

Show intensity scope window

Syntax

show(sIS)

Description

show(sIS) shows the display window of the phased.IntensityScope object, sIS.

Input Arguments

sIS — Intensity scope
phased.IntensityScope System object

Intensity scope, specified as a phased.IntensityScope System object.

Example: phased.IntensityScope

Examples

Hide and Show Intensity Scope

Create an angle-time-intensity scope. Use the phased.IntensityScope System object™ to display simulated intensity as a function of the angular motion of a moving target. After five steps in the processing loop, use the hide method to hide the scope. At completion of the loop, use the show method to show the scope.

Simulate data for 5 seconds with a time interval of 0.5 seconds between scan lines.

```
nsteps = 10;
dt = 0.5;
timespan = nsteps*dt;
```

Set Up IntensityScope System Object

Create an angle-time-intensity scope having azimuth angle bins spanning −180° to 180° with 1° resolution.

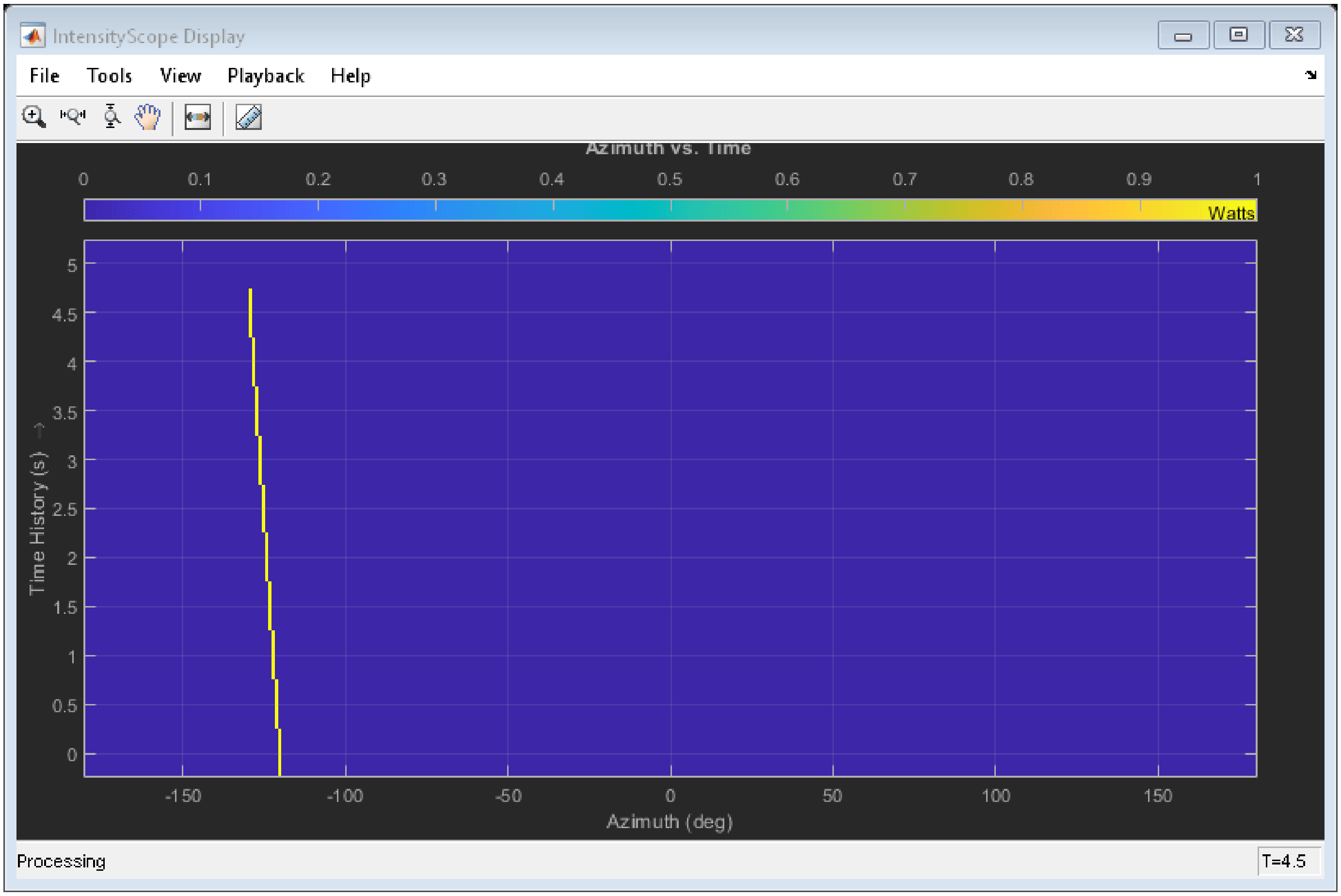
```
scanline = zeros(361,1);
angres = 1.0;
angmin = -180.0;
angmax = 180.0;
rtidisplay = phased.IntensityScope( ...
    'Name','IntensityScope Display',...
    'Title','Azimuth vs. Time',...
```

```
    'XLabel','Azimuth (deg)', ...
    'XResolution',angres,'XOffset',angmin,...
    'TimeResolution',dt,'TimeSpan',timespan, ...
    'IntensityUnits','Watts',...
    'Position',[100,100,800,450]);
```

Loop Over Scan Updates

Simulate angular motion and fill the bin containing the current angular position of the signal. Hide the scope after the 5th step and show the scope at the end of the simulation.

```
for k = 1:nsteps
    ang = -130.0 + k;
    binindexdx = floor((ang - angmin)/angres) + 1;
    scanline(binindexdx) = 1;
    rtidisplay(scanline);
    scanline(binindexdx) = 0;
    if k == 5
        hide(rtidisplay)
    end
    pause(.1);
end
show(rtidisplay)
```



Version History

Introduced in R2016a

step

System object: `phased.IntensityScope`
Package: `phased`

Update intensity scope display

Syntax

`step(sIS,data)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`step(sIS,data)` updates the intensity scope display with new scan lines from a real signal, `data`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`sIS` — Intensity scope display
`phased.IntensityScope` System object

Intensity scope display, specified as a `phased.IntensityScope` System object.

Example: `phased.IntensityScope`

`data` — Displayed intensity values
real-valued *N*-by-*M* matrix

Displayed intensity values, specified as a real-valued *N*-by-*M* matrix. The quantity *N* specifies the number of intensity bins in `data`. The quantity *M* specifies the number of intensity vectors in the data. Each column of the matrix creates a display line. Units are arbitrary. Specify the time interval between intensity vectors using the `TimeResolution` property.

Example: `[5.0;5.1;5.0;4.9]`

Data Types: `double`

Examples

RTI Display of Three Moving Targets

Use the `phased.IntensityScope` System object™ to display the intensities of the echoes of three moving targets as functions of range and time.

Create the Radar and Target System Objects

Set up the initial positions and velocities of the three targets. Use the `phased.Platform` System object to model radar and target motions. The radar is stationary while the targets undergo constant velocity motion. The simulation runs for 500 steps at 0.1 second increments, giving a total simulation time of 50 seconds.

```
nsteps = 500;
dt = .1;
timespan = nsteps*dt;
x1 = [60,0,0]';
x2 = [60,-80,40]';
x3 = [300,0,-300]';
v1 = [2,0,0]';
v2 = [10,5,6]';
v3 = [-10,2,-4]';
platform = phased.Platform([0,0,0]',[0,0,0]');
targets = phased.Platform([x1,x2,x3],[v1,v2,v3]);
```

Set Up Range Bins

Each echo is put into a range bin. The range bin resolution is 1 meter and the range is from 50 to 1000 meters.

```
rngres = 1.0;
rngmin = 50.0;
rngmax = 1000.0;
rngscan = [rngmin:rngres:rngmax];
```

Create the Gain Function

Define a range-dependent gain function to enhance the display of targets at larger ranges. The gain function amplifies the returned echo for visualization purposes only.

```
rangegain = @(rng)(1e12*rng^4);
```

Create the Intensity Scope

Set up the Intensity Scope using these properties.

- Use the `XResolution` property to set the width of each scan line bin to the range resolution of 1 km.
- Use the `XOffset` property to set the value of the lowest range bin to the minimum range of 50 km.
- Use the `TimeResolution` property to set the value of the scan line time difference to 0.1 s.
- Use the `TimeSpan` property to set the height of the display window to the time duration of the simulation.
- Use the `IntensityUnits` property to set the display units to `Watts`.

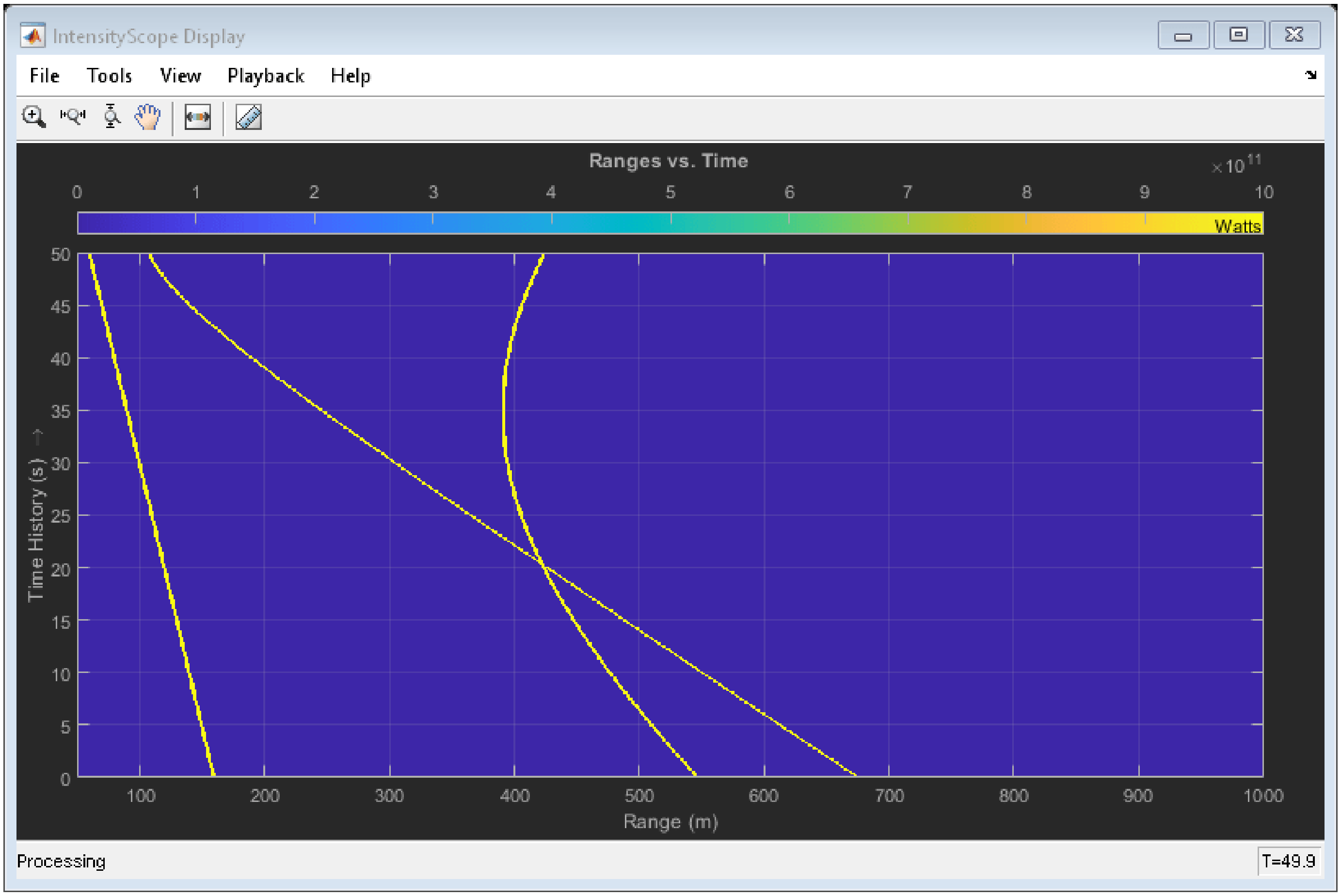
```
scope = phased.IntensityScope('Name','IntensityScope Display',...
    'Title','Ranges vs. Time','XLabel','Range (m)','XResolution',rngres,...
```

```
    'XOffset',rngmin,'TimeResolution',dt,'TimeSpan',timespan, ...
    'IntensityUnits','Watts','Position',[100,100,800,450]);
```

Run Simulation Loop

1. In this loop, move the targets at constant velocity using the `step` method of the `phased.Platform` System object.
2. Compute the target ranges using the `rangeangle` function.
3. Compute the target range bins by quantizing the range values in integer multiples of `rngres`.
4. Fill each target range bin and neighboring bins with a simulated radar intensity value.
5. Add the signal from each target to the scan line.
6. Call the `step` method of the `phased.IntensityScope` System object to display the scan lines.

```
for k = 1:nsteps
    xradar = platform(dt);
    xtgts = targets(dt);
    [rngs] = rangeangle(xtgts,xradar);
    scanline = zeros(size(rngscan));

    rngindx = ceil((rngs(1) - rngmin)/rngres);
    scanline(rngindx + [-1:1]) = rangegain(rngs(1))/(rngs(1)^4);

    rngindx = ceil((rngs(2) - rngmin)/rngres);
    scanline(rngindx + [-1:1]) = rangegain(rngs(2))/(rngs(2)^4);

    rngindx = ceil((rngs(3) - rngmin)/rngres);
    scanline(rngindx + [-1:1]) = rangegain(rngs(3))/(rngs(3)^4);

    scope(scanline.');
    pause(.1);
end
```



RTI and DTI Displays in Full Radar Simulation

Use the `phased.IntensityScope` System Object™ to display the detection output of a radar system simulation. The radar scenario contains a stationary single-element monostatic radar and three moving targets.

Set Radar Operating Parameters

Set the maximum range, peak power range resolution, operating frequency, transmitter gain, and target radar cross-section.

```
max_range = 5000;
range_res = 50;
fc = 10e9;
tx_gain = 20;
peak_power = 5500.0;
```

Choose the signal propagation speed to be the speed of light, and compute the signal wavelength corresponding to the operating frequency.

```
c = physconst('LightSpeed');
lambda = c/fc;
```

Compute the pulse bandwidth from the range resolution. Set the sampling rate, `fs`, to twice the pulse bandwidth. The noise bandwidth is also set to the pulse bandwidth. The radar integrates a number of pulses set by `num_pulse_int`. The duration of each pulse is the inverse of the pulse bandwidth.

```
pulse_bw = c/(2*range_res);
pulse_length = 1/pulse_bw;
fs = 2*pulse_bw;
noise_bw = pulse_bw;
num_pulse_int = 10;
```

Set the pulse repetition frequency to match the maximum range of the radar.

```
prf = c/(2*max_range);
```

Create System Objects for the Model

Choose a rectangular waveform.

```
waveform = phased.RectangularWaveform('PulseWidth',pulse_length,...
    'PRF',prf,'SampleRate',fs);
```

Set the receiver amplifier characteristics.

```
amplifier = phased.ReceiverPreamp('Gain',20,'NoiseFigure',0,...
    'SampleRate',fs,'EnableInputPort',true,'SeedSource','Property',...
    'Seed',2007);
transmitter = phased.Transmitter('Gain',tx_gain,'PeakPower',peak_power,...
    'InUseOutputPort',true);
```

Specify the radar antenna as a single isotropic antenna.

```
antenna = phased.IsotropicAntennaElement('FrequencyRange',[5e9 15e9]);
```

Set up a monostatic radar platform.

```
radarplatform = phased.Platform('InitialPosition',[0; 0; 0],...
    'Velocity',[0; 0; 0]);
```

Set up the three target platforms using a single System object.

```
targetplatforms = phased.Platform(...
    'InitialPosition',[2000.66 3532.63 3845.04; 0 0 0; 0 0 0], ...
    'Velocity',[150 -150 0; 0 0 0; 0 0 0]);
```

Create the radiator and collector System objects.

```
radiator = phased.Radiator('Sensor',antenna,'OperatingFrequency',fc);
collector = phased.Collector('Sensor',antenna,'OperatingFrequency',fc);
```

Set up the three target RCS properties.

```
targets = phased.RadarTarget('MeanRCS',[1.6 2.2 1.05],'OperatingFrequency',fc);
```

Create System object to model two-way freespace propagation.

```
channels= phased.FreeSpace('SampleRate',fs,'TwoWayPropagation',true,...
    'OperatingFrequency',fc);
```

Define a matched filter.

```
MFcoef = getMatchedFilter(waveform);
filter = phased.MatchedFilter('Coefficients',MFcoef,'GainOutputPort',true);
```

Create Range and Doppler Bins

Set up the fast-time grid. Fast time is the sampling time of the echoed pulse relative to the pulse transmission time. The range bins are the ranges corresponding to each bin of the fast time grid.

```
fast_time = unigrid(0,1/fs,1/prf,'[)');
range_bins = c*fast_time/2;
```

To compensate for range loss, create a time varying gain System Object.

```
gain = phased.TimeVaryingGain('RangeLoss',2*fspl(range_bins,lambda),...
    'ReferenceLoss',2*fspl(max_range,lambda));
```

Set up Doppler bins. Doppler bins are determined by the pulse repetition frequency. Create an FFT System object for Doppler processing.

```
DopplerFFTbins = 32;
DopplerRes = prf/DopplerFFTbins;
fft = dsp.FFT('FFTLengthSource','Property',...
    'FFTLength',DopplerFFTbins);
```

Create Data Cube

Set up a reduced data cube. Normally, a data cube has fast-time and slow-time dimensions and the number of sensors. Because the data cube has only one sensor, it is two-dimensional.

```
rx_pulses = zeros(numel(fast_time),num_pulse_int);
```

Create IntensityScope System Objects

Create two `IntensityScope` System objects, one for Doppler-time-intensity and the other for range-time-intensity.

```
dtiscope = phased.IntensityScope('Name','Doppler-Time Display',...
    'XLabel','Velocity (m/sec)', ...
    'XResolution',dop2speed(DopplerRes,c/fc)/2, ...
    'XOffset',dop2speed(-prf/2,c/fc)/2,...
    'TimeResolution',0.05,'TimeSpan',5,'IntensityUnits','Mag');
rtiscope = phased.IntensityScope('Name','Range-Time Display',...
    'XLabel','Range (m)', ...
    'XResolution',c/(2*fs), ...
    'TimeResolution',0.05,'TimeSpan',5,'IntensityUnits','Mag');
```

Run the Simulation Loop over Multiple Radar Transmissions

Transmit 2000 pulses. Coherently process groups of 10 pulses at a time.

For each pulse:

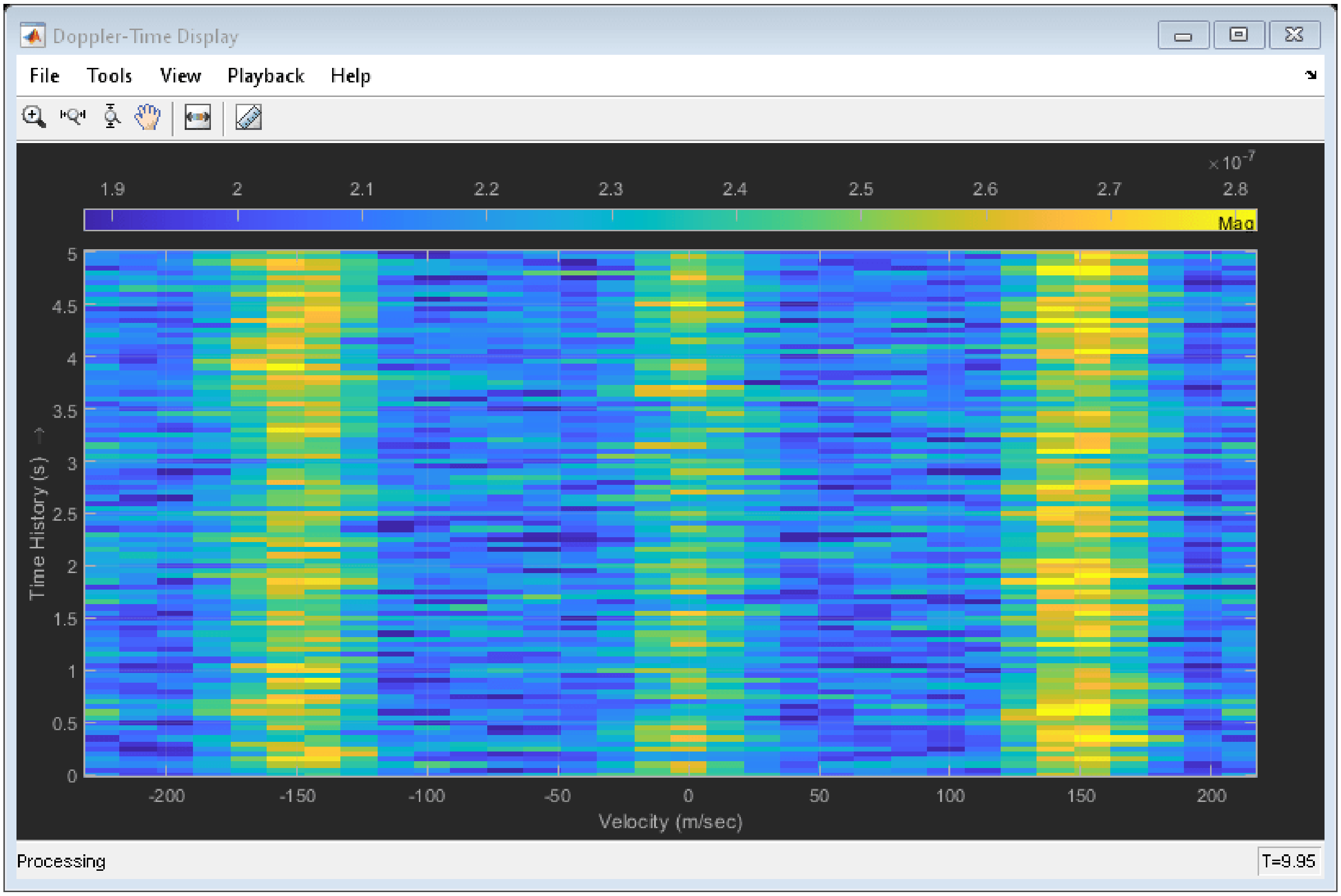
1. Update the radar position and velocity `radarplatform`
2. Update the target positions and velocities `targetplatforms`
3. Create the pulses of a single wave train to be transmitted `transmitter`
4. Compute the ranges and angles of the targets with respect to the radar

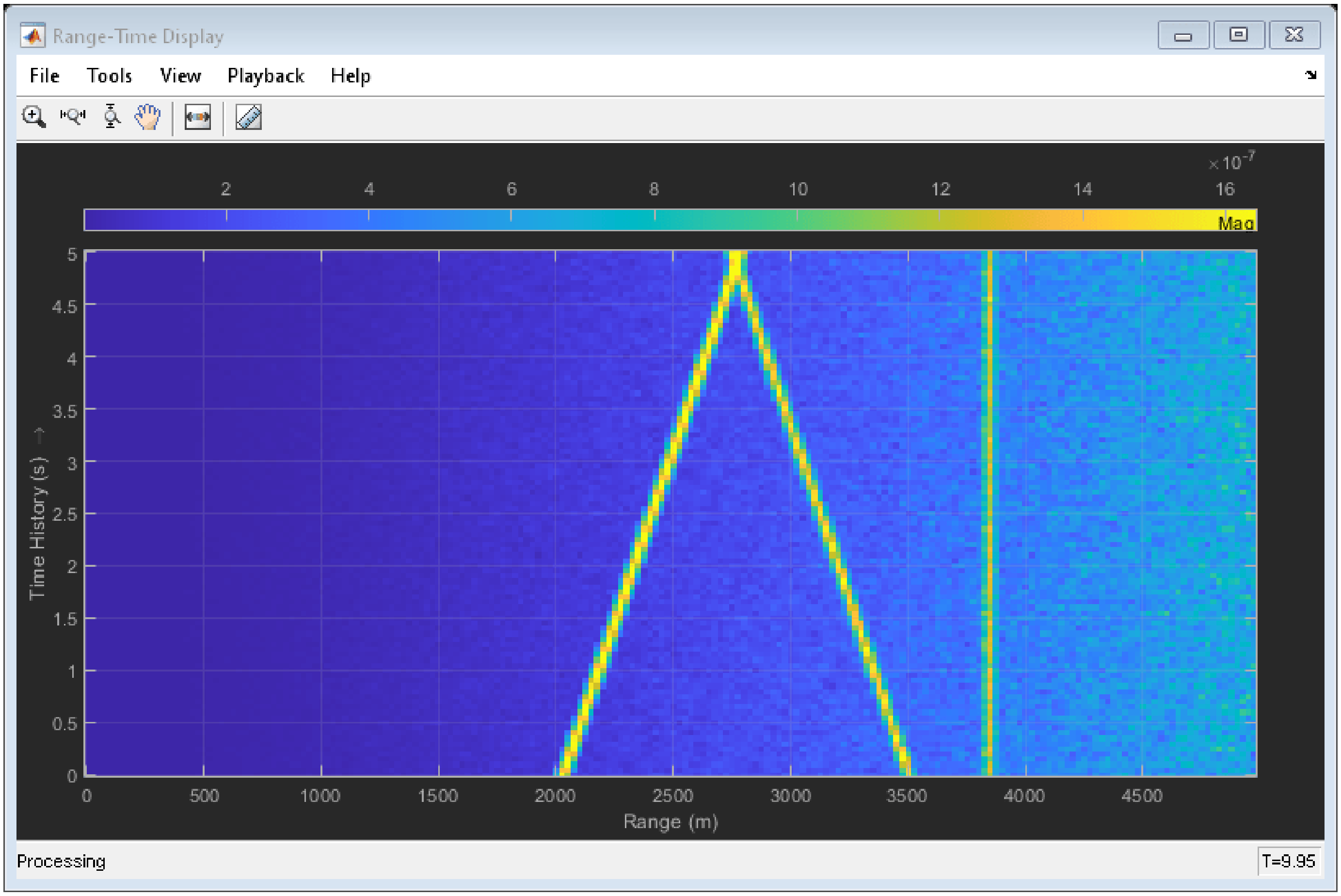
5 Radiate the signals to the targets `radiator`
6 Propagate the pulses to the target and back `channels`
7 Reflect the signals off the target `targets`
8 Receive the signal `sCollector`
9 Amplify the received signal `amplifier`
10 Form data cube

For each set of 10 pulses in the data cube:

1 Match filter each row (fast-time dimension) of the data cube.
2 Compute the Doppler shifts for each row (slow-time dimension) of the data cube.

```
pri = 1/prf;
nsteps = 200;
for k = 1:nsteps
    for m = 1:num_pulse_int
        [ant_pos,ant_vel] = radarplatform(pri);
        [tgt_pos,tgt_vel] = targetplatforms(pri);
        sig = waveform();
        [s,tx_status] = transmitter(sig);
        [~,tgt_ang] = rangeangle(tgt_pos,ant_pos);
        tsig = radiator(s,tgt_ang);
        tsig = channels(tsig,ant_pos,tgt_pos,ant_vel,tgt_vel);
        rsig = targets(tsig);
        rsig = collector(rsig,tgt_ang);
        rx_pulses(:,m) = amplifier(rsig,~(tx_status>0));
    end

    rx_pulses = filter(rx_pulses);
    MFdelay = size(MFcoef,1) - 1;
    rx_pulses = buffer(rx_pulses((MFdelay + 1):end), size(rx_pulses,1));
    rx_pulses = gain(rx_pulses);
    range = pulsint(rx_pulses,'noncoherent');
    rtiscope(range);
    dshift = fft(rx_pulses.');
    dshift = fftshift(abs(dshift),1);
    dtiscope(mean(dshift,2));
    radarplatform(.05);
    targetplatforms(.05);
end
```


Doppler-Time Display
File
Tools
View
Playback
Help
×10⁻⁷
1.9
2
2.1
2.2
2.3
2.4
2.5
2.6
2.7
2.8
Mag
Time History (s)
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-200
-150
-100
-50
0
50
100
150
200
Velocity (m/sec)
Processing
T=9.95



All of the targets lie on the x-axis. Two targets are moving along the x-axis and one is stationary. Because the radar is at the origin, you can read the target speed directly from the Doppler-Time Display window. The values agree with the specified velocities of -150, 150, and 0 m/sec.

Intensity Scope Display of Target Angular Motion

Use the `phased.IntensityScope` System object™ to display the angular motions of moving targets as functions of time. Each horizontal line (scan line) shows the strength of radar echoes at different azimuth angles. Azimuth space is divided into azimuth bins and each bin is filled with a simulated value depending upon the position of the targets.

Create Radar and Target System Objects

Set up the initial positions and velocities of the three targets. Use the `phased.Platform` System object to model radar and target motions. The radar is stationary while the targets undergo constant velocity motion. The simulation runs for 200 steps at 0.5 second intervals, giving a total simulation time of 100 seconds.

```
nsteps = 200;
dt = 0.5;
timespan = nsteps*dt;
x1 = [60,0,0]';
```

```
x2 = [60,-80,40]';
x3 = [300,0,-300]';
x3 = [-300,0,-300]';

v1 = [2,0,0]';
v2 = [10,5,6]';
v3 = [-10,2,-4]';
radarplatform = phased.Platform([0,0,0]',[0,0,0]');
targets = phased.Platform([x1,x2,x3],[v1,v2,v3]);
```

Set Up Azimuth Angle Bins

The signal for each echo is put into an angle bin and two adjacent bins. Bin resolution is 1 degree and the angle span is from −180 to 180 degrees.

```
angres = 1.0;
angmin = -180.0;
angmax = 180.0;
angscan = [angmin:angres:angmax];
na = length(angscan);
```

Range Gain Function

Define a range-dependent gain function to enhance the display of targets at larger ranges. The gain function amplifies the returned echo for visualization purposes only.

```
rangegain = @(rng)(1e12*rng^4);
```

Set Up Scope Viewer

The `XResolution` name-value pair specifies the width of each bin of the scan line. The `XOffset` sets the value of the lowest azimuth angle bin. The `TimeResolution` name-value pair specifies the time difference between scan lines. The `TimeSpan` name-value pair sets the height of the display window. A scan line is created with each call to the `step` method. Intensity units are amplitude units.

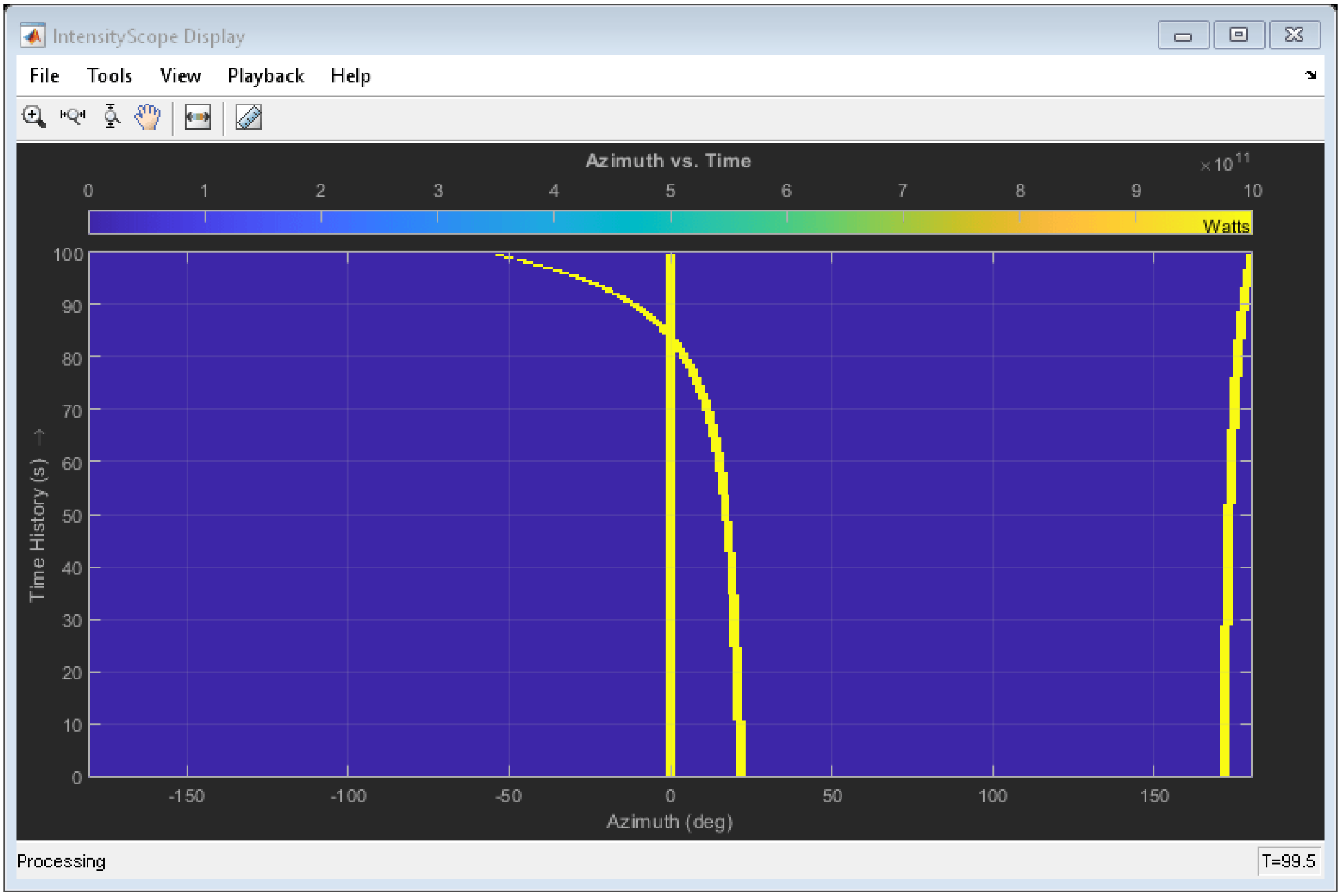
```
scope = phased.IntensityScope( ...
    'Name','IntensityScope Display',...
    'Title','Azimuth vs. Time',...
    'XLabel','Azimuth (deg)', ...
    'XResolution',angres,'XOffset',angmin,...
    'TimeResolution',dt,'TimeSpan',timespan, ...
    'IntensityUnits','Watts',...
    'Position',[100,100,800,450]);
```

Update-Display Loop

1. In this loop, move the targets at constant velocity using the `step` method of the `phased.Platform` System object.
2. Compute the target ranges and azimuth angles using the `rangeangle` function.
3. Compute the azimuth angle bins by quantizing the azimuth angle values in integer multiples of `angres`.
4. Fill each target azimuth bin and neighboring bins with a simulated radar intensity value.
5. Call the `phased.IntensityScope` `step` method to display the scan line.

```
for k = 1:nsteps
    xradar = radarplatform(dt);
```

```
    xtgts = targets(dt);
    [rngs,angs] = rangeangle(xtgts,xradar);
    scanline = zeros(size(angscan));

    angindx = ceil((angs(1,1) - angmin)/angres) + 1;
    idx = angindx + [-1:1];
    idx(idx>na)=[];
    idx(idx<1)=[];
    scanline(idx) = rangegain(rngs(1))/(rngs(1)^4);

    angindx = ceil((angs(1,2) - angmin)/angres) + 1;
    idx = angindx + [-1:1];
    idx(idx>na)=[];
    idx(idx<1)=[];
    scanline(idx) = rangegain(rngs(2))/(rngs(2)^4);

    angindx = ceil((angs(1,3) - angmin)/angres) + 1;
    idx = angindx + [-1:1];
    idx(idx>na)=[];
    idx(idx<1)=[];
    scanline(idx) = rangegain(rngs(3))/(rngs(3)^4);
    scope(scanline.');
    pause(.1);
end
```



Version History

Introduced in R2016a

phased.IsoSpeedUnderwaterPaths

Package: `phased`

Isospeed multipath sonar channel

Description

The `phased.IsoSpeedUnderwaterPaths` System object creates an underwater acoustic channel to propagate narrowband sound from point to point. The channel has finite constant depth with air-water and water-bottom interfaces. Both interfaces are planar and horizontal. Sound speed is constant throughout the channel. The object generates multiple propagation paths in the channel using the acoustical method of images (see [3]). Because sound speed is constant, all propagation paths are straight lines between the source, boundaries, and receiver. There is always one direct line-of-sight path. For each propagation path, the object outputs range-dependent time delay, gain, Doppler factor, reflection loss, and spreading loss. You can use the channel data as input to the multipath sound propagator, `phased.MultipathChannel`.

To model an isospeed channel :

1. Define and set up the channel. You can set `phased.IsoSpeedUnderwaterPaths` System object properties at construction time or leave them to their default values. See “Construction” on page 1-733. Some properties that you set at construction time can be changed later. These properties are *tunable*.
2. To create the multipath channel, call the `step` method of `phased.IsoSpeedUnderwaterPaths`. The output of the method depends on the properties of the `phased.IsoSpeedUnderwaterPaths` System object. You can change tunable properties at any time.

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`channel = phased.IsoSpeedUnderwaterPaths` creates an isospeed multipath underwater channel System object, `channel`.

`channel = phased.IsoSpeedUnderwaterPaths(Name,Value)` creates an isospeed multipath underwater channel System object, `channel`, with each specified property `Name` set to the specified `Value`. You can specify additional name and value pair arguments in any order as (`Name1,Value1`,...,`NameN,ValueN`).

Properties

`ChannelDepth` — Channel depth

`100` (default) | positive scalar

Channel depth, specified as a positive scalar. Units are in meters.

Example: `250.0`

Data Types: `double`

`PropagationSpeed` — Underwater sound propagation speed
`1520.0` (default) | positive scalar

Underwater sound propagation speed, specified as a positive scalar. Units are in meter per second. The default value is a commonly-used underwater sound propagation speed.

Example: `1502.0`

Data Types: `double`

`NumPathsSource` — Source of number of propagation paths
`'Auto'` (default) | `'Property'`

The source of the number of propagation paths, specified as `'Auto'` or `'Property'`. If you set this property to `'Auto'`, the object automatically determines the number of paths based on spreading and reflection losses. If you set this property to `'Property'`, you specify the number of paths using the `NumPaths` property.

When `NumPathsSource` is set to `'Auto'`, only paths having a total loss greater than 20 dB below the direct path loss are returned.

Example: `'Property'`

Data Types: `char`

`NumPaths` — Number of propagation paths
`10` (default) | positive integer

The number of propagation paths, specified as a positive integer between 1 and 51, inclusive.

Example: `11`

Dependencies

To enable this property, set the `NumPathsSource` property to `'Property'`.

Data Types: `double`

`CoherenceTime` — Channel coherence time
`0` (default) | nonnegative scalar

Channel coherence time, specified as a nonnegative scalar. Coherence time is a measure of the temporal stability of the channel. The object keeps a record of cumulative step time. When the cumulative step time exceeds the coherence time, propagation paths are recomputed and the cumulative step time is reset to zero. If you set this quantity to zero, the propagation paths are update at each call to `step`. Units are in seconds.

Example: `5.0`

Data Types: `double`

`BottomLoss` — Bottom reflection loss
`6` (default) | nonnegative scalar

Bottom reflection loss, specified as a nonnegative scalar. This value applies to each bottom reflection of a path. Units are in dB.

Example: `10`

Data Types: `double`

`LossFrequencies` — Absorption loss frequencies
`[1:100]*1000` (default) | positive real-valued vector

Frequencies for which to compute absorption loss, specified as a positive real-valued vector. Units are in Hz.

Example: `[1000:100:3000]`

Data Types: `double`

`TwoWayPropagation` — Enable two-way propagation
`false` (default) | `true`

Enable two-way propagation, specified as a `false` or `true`. Set this property to `true` to perform round-trip propagation between the signal origin and destination specified in `step`. Set this property to `false` to perform only one-way propagation from the origin to the destination.

Example: `true`

Data Types: `logical`

Methods

step	Create propagation paths in an isospeed multipath sound channel
reset	Reset state of System object

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Create One-Way Multipath Underwater Sound Channel

Create a 5-path underwater sound channel and display the propagation path matrix. Assume the source is stationary and the receiver is moving along the *x*-axis towards the source at 20 kph. Assume one-way propagation.

```
speed = -20*1000/3600;
numpaths = 5;
channelpaths = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',10, ...
    'NumPathsSource','Property','NumPaths',numpaths,'CoherenceTime',5);
tstep = 1;
srcpos = [0;0;-160];
rcvpos = [500;0;-50];
srcvel = [0;0;0];
rcvvel = [speed;0;0];
pathmat = channelpaths(srcpos,rcvpos,srcvel,rcvvel,tstep);
disp(pathmat)
```

```
    0.3356    0.3556    0.4687    0.3507    0.3791
    1.0000   -1.0000   -0.3162    0.3162   -0.3162
   54.1847   54.6850   57.0766   54.5652   55.2388
```

The first row contains the time delay in seconds. The second row contains the bottom reflection loss coefficients, and the third row contains the spreading loss in dB. The reflection loss coefficient for the first path is 1.0 because the direct path has no boundary reflections. The reflection loss coefficient for the second path is -1.0 because the path has only a surface reflection.

Create Two-Way Multipath Underwater Sound Channel

Create a 7-path underwater sound channel and display the propagation path matrix. Assume the source is stationary and the target is moving along the *x*-axis towards the source at 20 kph. Assume two-way propagation.

```
speed = -20*1000/3600;
numpaths = 7;
channelpaths = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',10, ...
    'NumPathsSource','Property','NumPaths',numpaths,'CoherenceTime',5,...
    'TwoWayPropagation',true);
tstep = 1;
srcpos = [0;0;-160];
tgtpos = [500;0;-50];
srcvel = [0;0;0];
tgtvel = [speed;0;0];
[pathmat,dop,aloss,tgtangs,srcangs] = channelpaths(srcpos,tgtpos,srcvel,tgtvel,tstep);
disp(pathmat)
```

```
    0.6712    0.7112    0.9374    1.0354    0.7014    0.7581    1.0152
    1.0000    1.0000    0.1000    0.1000    0.1000    0.1000    0.0100
  108.3693  109.3699  114.1531  115.8772  109.1304  110.4775  115.5355
```

The first row contains the time delay in seconds. The second row contains the bottom reflection loss coefficients, and the third row contains the spreading loss in dB. The reflection loss coefficient for the first path is 1.0 because the direct path has no boundary reflections. The reflection loss coefficient for the second path is -1.0 because the path has only a surface reflection.

Propagate Sound in Channel Having Unknown Number of Paths

Create an underwater sound channel and plot the combined received signal. Automatically find the number of paths. Assume that the source is stationary and that the receiver is moving along the *x*-axis toward the source at 20 km/h. Assume the default one-way propagation.

```
speed = -20*1000/3600;
channel = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',5, ...
    'NumPathsSource','Auto','CoherenceTime',5);
tstep = 1;
srcpos = [0;0;-160];
rcvpos = [500;0;-50];
srcvel = [0;0;0];
rcvvel = [speed;0;0];
```

Compute the path matrix, Doppler factor, and losses. The propagator outputs 51 paths output but some paths can contain `Nan` values.

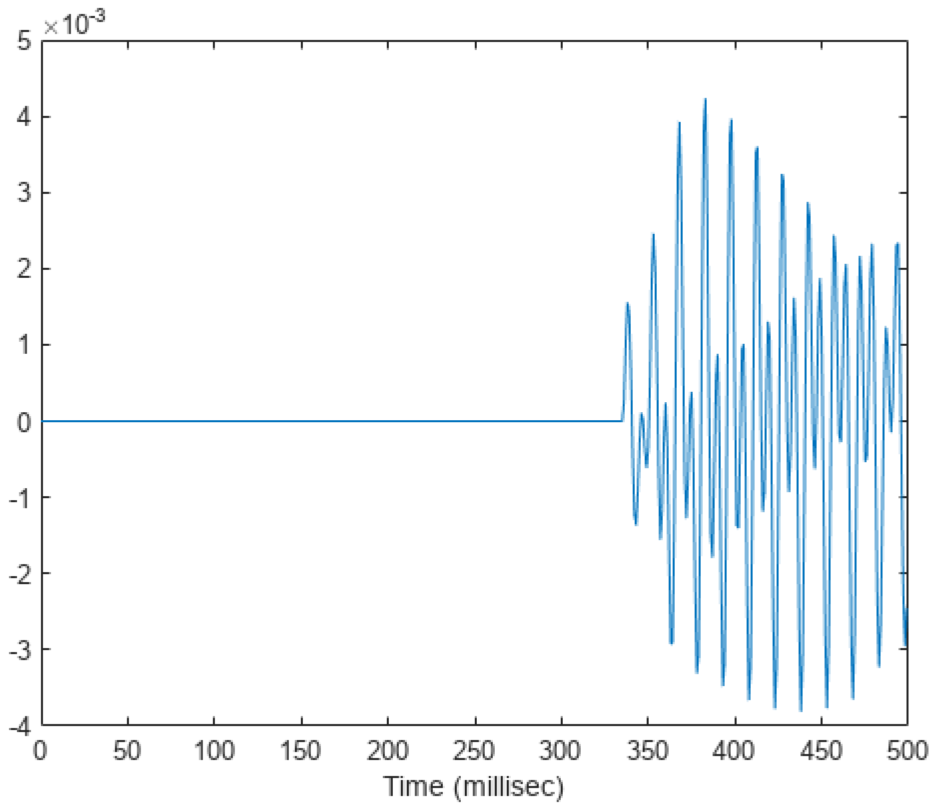
```
[pathmat,dop,absloss,rcvangs,srcangs] = channel(srcpos,rcvpos,srcvel,rcvvel,tstep);
```

Create of a 100 Hz signal with 500 samples. Assume that all the paths have the same signal. Use a `phased.MultipathChannel` System object™ to propagate the signals to the receiver. `phased.MultipathChannel` accepts as input all paths produced by `phased.IsoSpeedUnderwaterPaths` but ignores paths that have `NaN` values.

```
fs = 1e3;
nsamp = 500;
propagator = phased.MultipathChannel('OperatingFrequency',10e3,'SampleRate',fs);
t = [0:(nsamp-1)]'/fs;
sig0 = sin(2*pi*100*t);
numpaths = size(pathmat,2);
sig = repmat(sig0,1,numpaths);
propsig = propagator(sig,pathmat,dop,absloss);
```

Plot the real part of the coherent sum of the propagated signals.

```
plot(t*1000,real(sum(propsig,2)))
xlabel('Time (millisec)')
```



Doppler Stretching of Sonar Signal

Compare the duration of a propagated signal from a stationary sonar to that of a moving sonar. The moving sonar has a radial velocity of 25 m/s away from the target. In each case, propagate the signal along a single path. Assume one-way propagation.

Define the sonar system parameters: maximum unambiguous range, required range resolution, operating frequency, and propagation speed.

```
maxrange = 5000.0;
rngres = 10.0;
fc = 20.0e3;
csound = 1520.0;
```

Use a rectangular waveform for the transmitted signal.

```
prf = csound/(2*maxrange);
pulseWidth = 8*rngres/csound;
pulseBW = 1/pulseWidth;
fs = 80*pulseBW;
waveform = phased.RectangularWaveform('PulseWidth',pulseWidth,'PRF',prf, ...
    'SampleRate',fs);
```

Specify the sonar positions.

```
sonarplatform1 = phased.Platform('InitialPosition',[0;0;-60],'Velocity',[0;0;0]);
sonarplatform2 = phased.Platform('InitialPosition',[0;0;-60],'Velocity',[0;-25;0]);
```

Specify the target position.

```
targetplatform = phased.Platform('InitialPosition',[0;500;-60],'Velocity',[0;0;0]);
```

Define the underwater path and propagation channel objects.

```
paths = phased.IsoSpeedUnderwaterPaths('ChannelDepth',100, ...
    'CoherenceTime',0,'NumPathsSource','Property','NumPaths',1, ...
    'PropagationSpeed',csound);
propagator = phased.MultipathChannel('SampleRate',fs,'OperatingFrequency',fc);
```

Create the transmitted waveform.

```
wav = waveform();
nsamp = size(wav,1);
rxpulses = zeros(nsamp,2);
t = (0:nsamp-1)/fs;
```

Transmit the signal and then receive the echo at the stationary sonar.

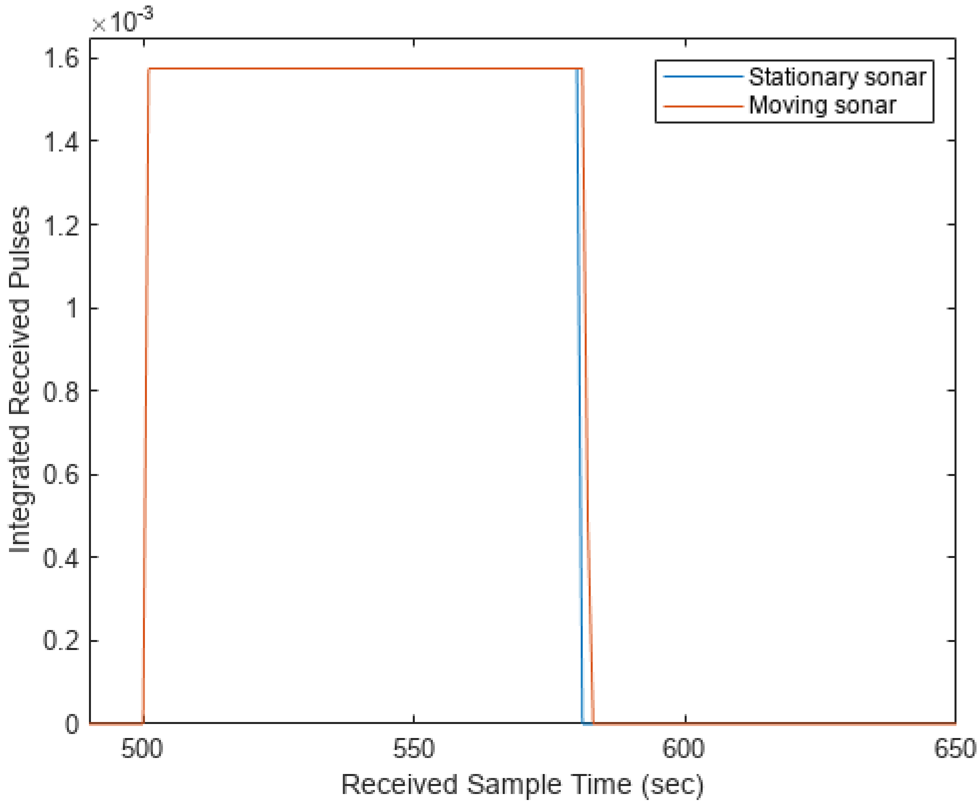
```
[pathmat,dop,aloss,~,~] = paths(sonarplatform1.InitialPosition, ...
    targetplatform.InitialPosition,sonarplatform1.InitialVelocity, ...
    targetplatform.InitialVelocity,1/prf);
rxpulses(:,1) = propagator(wav,pathmat,dop,aloss);
```

Transmit and receive at the moving sonar.

```
[pathmat,dop,aloss,~,~] = paths(sonarplatform2.InitialPosition, ...
    targetplatform.InitialPosition,sonarplatform2.Velocity, ...
    targetplatform.Velocity,1/prf);
rxpulses(:,2) = propagator(wav,pathmat,dop,aloss);
```

Plot the received pulses.

```
plot(abs(rxpulses))
xlim([490 650])
ylim([0 1.65e-3])
legend('Stationary sonar','Moving sonar')
xlabel('Received Sample Time (sec)')
ylabel('Integrated Received Pulses')
```



The signal received at the moving sonar has increased in duration compared to the stationary sonar.

Version History

Introduced in R2017a

References

[1] Urick, R.J. *Principles of Underwater Sound, 3rd Edition*. New York: Peninsula Publishing, 1996.

[2] Sherman, C.S. and J.Butler *Transducers and Arrays for Underwater Sound*. New York: Springer, 2007.

[3] Allen, J.B. and D. Berkely, “Image method for efficiently simulating small-room acoustics”, J. Acoust. Soc. Am, Vol 65, No. 4. April 1979.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects

`phased.BackscatterSonarTarget` | `phased.IsotropicHydrophone` | `phased.IsotropicProjector` | `phased.MultipathChannel`

Topics

"Underwater Target Detection with an Active Sonar System"
"Locating an Acoustic Beacon with a Passive Sonar System"

step

System object: `phased.IsoSpeedUnderwaterPaths`
Package: `phased`

Create propagation paths in an isospeed multipath sound channel

Syntax

```
pathmat = step(channel,srcpos,destpos,srcvel,destvel,T)
[pathmat,dop,aloss,destang,srcang] = step(channel,srcpos,destpos,srcvel,
destvel,T)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`pathmat = step(channel,srcpos,destpos,srcvel,destvel,T)` returns the propagation paths matrix, `pathmat`, for a multipath underwater acoustic channel. The matrix describes one or two-way propagation from the signal source position, `srcpos`, to the signal destination position, `destpos`. The velocity of the signal source is specified in `srcvel` and the velocity of the signal destination is specified in `destvel`. `T` is the step time interval.

When you use this method for one-way propagation, `srcpos` refers to the origin of the signal and `destpos` to the receiver. One-way propagation modeling is useful for passive sonar and underwater communications.

When you use this method for two-way propagation, `destpos` now refers to the reflecting target, not the sonar receiver. A two-way path consists of a one-way path from source to target and then along an identical one-way path from target to receiver (which is collocated with the source). Two-way propagation modeling is useful for active sonar systems.

`[pathmat,dop,aloss,destang,srcang] = step(channel,srcpos,destpos,srcvel,destvel,T)` also returns the Doppler factor, `dop`, the frequency dependent absorption loss, `aloss`, the receiver arrival angles, `destang`, and the `srcang` transmitting angles.

When you use this method for two-way propagation, `destang` now refers to the reflecting target, not the sonar receiver.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Input Arguments

`channel` — Isospeed underwater channel path
`phased.IsoSpeedUnderwaterPaths` System object

Isospeed underwater channel paths, specified as a `phased.IsoSpeedUnderwaterPaths` System object.

Example: `phased.IsoSpeedUnderwaterPaths`

`srcpos` — Source of sonar signals
real-valued 3-by-1 column vector

Source of the sonar signal, specified as a real-valued 3-by-1 column vector. Position units are meters.

Example: `[1000;100;500]`

Data Types: `double`

`destpos` — Destination of signal
real-valued 3-by-1 column vector

Destination position of the signal, specified as a real-valued 3-by-1 column vector. Position units are in meters.

Example: `[0;0;0]`

Data Types: `double`

`srcvel` — Velocity of signal source
real-valued 3-by-1 column vector

Velocity of signal source, specified as a real-valued 3-by-1 column vector. Velocity units are in meters per second.

Example: `[10;0;5]`

Data Types: `double`

`destvel` — Velocity of signal destination
real-valued 3-by-1 column vector

Velocity of signal destination, specified as a real-valued 3-by-1 column vector. Velocity units are in meters per second.

Example: `[0;0;0]`

Data Types: `double`

`T` — Elapsed time of current step
positive scalar

Elapsed time of current step, specified as a positive scalar. Time units are in seconds.

Example: `0.1`

Data Types: `double`

Output Arguments

`pathmat` — Propagation paths matrix
real-valued 3-by-*N* matrix

Propagation paths matrix, returned as a real-valued 3-by-*N* matrix. *N* is the number of paths in the channel. Each column represents a path. When you set `NumPathsSource` to `'Auto'`, *N* is 51. In this case, any columns filled with `NaN` do not correspond to found paths. The matrix rows represent:

Row	Data
1	Propagation delays for each path. Units are in seconds.
2	Total reflection coefficient for each path. Units are dimensionless
3	Spreading loss for each path. Units are in dB.

Except for the direct path, paths consists of alternating surface and bottom reflections. The losses for multiple reflections are multiplied. Bottom loss per reflection is specified by the `BottomLoss` property. The loss at the surface is *–1* indicating no loss, but only a 180° phase change. This is because the air-water interface surface is a pressure-release surface.

Data Types: `double`

`dop` — Doppler factor
real-valued *N*-by-1 row vector

Doppler factor, returned as a real-valued *N*-by-1 row vector where *N* is the number of paths. The Doppler factor multiplies the transmitted frequency to produce the Doppler-shifted received frequency for each path. The Doppler shift is defined as the difference between the transmitted frequency and the received frequency. The Doppler factor also defines the time compression or expansion of a signal. Units are dimensionless.

Data Types: `double`

`aloss` — Frequency-dependent absorption loss
real-valued *K*-by-*(N+1)* matrix

Frequency-dependent absorption loss, returned as a real-valued *K*-by-*(N+1)* matrix. *K* is the number of frequencies specified in the `LossFrequencies` property. *N* is the number of paths returned. The first column of `aloss` contains the absorption loss frequencies in Hz. You specify the frequencies using the `LossFrequencies` property. The remaining columns contain the absorption losses for each frequency. There is one column for each path. Units are in dB.

Data Types: `double`

`destang` — Angles of paths at destination
real-valued 2-by-*N* matrix

Angles of paths at destination, returned as a real-valued 2-by-*N* matrix. Each column contains the direction of the received path with respect to the destination position as azimuth and elevation, `[az;el]`. Units are in degrees.

Data Types: `double`

`srcang` — Angles of paths from source
real-valued 2-by-*N* matrix

Angles of paths from source, returned as a real-valued 2-by-*N* matrix. Each column contains the direction of the transmitted path with respect to the source position as azimuth and elevation, `[az;el]`. Units are in degrees.

Data Types: `double`

Examples

Create One-Way Multipath Underwater Sound Channel

Create a 5-path underwater sound channel and display the propagation path matrix. Assume the source is stationary and the receiver is moving along the *x*-axis towards the source at 20 kph. Assume one-way propagation.

```
speed = -20*1000/3600;
numpaths = 5;
channelpaths = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',10, ...
    'NumPathsSource','Property','NumPaths',numpaths,'CoherenceTime',5);
tstep = 1;
srcpos = [0;0;-160];
rcvpos = [500;0;-50];
srcvel = [0;0;0];
rcvvel = [speed;0;0];
pathmat = channelpaths(srcpos,rcvpos,srcvel,rcvvel,tstep);
disp(pathmat)
```

```
    0.3356    0.3556    0.4687    0.3507    0.3791
    1.0000   -1.0000   -0.3162    0.3162   -0.3162
   54.1847   54.6850   57.0766   54.5652   55.2388
```

The first row contains the time delay in seconds. The second row contains the bottom reflection loss coefficients, and the third row contains the spreading loss in dB. The reflection loss coefficient for the first path is 1.0 because the direct path has no boundary reflections. The reflection loss coefficient for the second path is -1.0 because the path has only a surface reflection.

Create Two-Way Multipath Underwater Sound Channel

Create a 7-path underwater sound channel and display the propagation path matrix. Assume the source is stationary and the target is moving along the *x*-axis towards the source at 20 kph. Assume two-way propagation.

```
speed = -20*1000/3600;
numpaths = 7;
channelpaths = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',10, ...
    'NumPathsSource','Property','NumPaths',numpaths,'CoherenceTime',5,...
    'TwoWayPropagation',true);
tstep = 1;
srcpos = [0;0;-160];
tgtpos = [500;0;-50];
srcvel = [0;0;0];
```

```
tgtvel = [speed;0;0];
[pathmat,dop,aloss,tgtangs,srcangs] = channelpaths(srcpos,tgtpos,srcvel,tgtvel,tstep);
disp(pathmat)
```

```
    0.6712    0.7112    0.9374    1.0354    0.7014    0.7581    1.0152
    1.0000    1.0000    0.1000    0.1000    0.1000    0.1000    0.0100
  108.3693  109.3699  114.1531  115.8772  109.1304  110.4775  115.5355
```

The first row contains the time delay in seconds. The second row contains the bottom reflection loss coefficients, and the third row contains the spreading loss in dB. The reflection loss coefficient for the first path is 1.0 because the direct path has no boundary reflections. The reflection loss coefficient for the second path is -1.0 because the path has only a surface reflection.

Automatically Find the Number of Paths

Create an underwater sound channel and display the propagation paths which are found automatically. Assume the source is stationary and the receiver is moving along the *x*-axis towards the source at 20 kph. Assume two-way propagation.

```
speed = -20*1000/3600;
channelpaths = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',5, ...
    'NumPathsSource','Auto','CoherenceTime',5,'TwoWayPropagation',true);
tstep = 1;
srcpos = [0;0;-160];
tgtpos = [500;0;-50];
srcvel = [0;0;0];
tgtpos = [speed;0;0];
[pathmat,dop,aloss,rcvangs,srcangs] = channelpaths(srcpos,tgtpos,srcvel,tgtpos,tstep);
```

Display the first 7 columns of `pathmat`. Some columns are filled with `NaN`s.

```
disp(pathmat(:,1:7))
```

```
    0.2107    0.2107       NaN       NaN       NaN       NaN       NaN
    1.0000    1.0000       NaN       NaN       NaN       NaN       NaN
   88.1753   88.1753       NaN       NaN       NaN       NaN       NaN
```

Select the column indices of the valid paths from the entire matrix.

```
idx = find(~isnan(pathmat(1,:)))
```

```
idx = 1×4

     1     2    27    28
```

Display the valid paths information.

```
validpaths = pathmat(:,idx)
```

```
validpaths = 3×4

    0.2107    0.2107    0.3159    0.3159
    1.0000    1.0000    0.3162    0.3162
   88.1753   88.1753   95.2131   95.2131
```

The first row contains the time delays in seconds. The second row contains the bottom reflected loss coefficients, and the third row contains the spreading losses.

Version History

Introduced in R2017a

reset

System object: `phased.IsoSpeedUnderwaterPaths`
Package: `phased`

Reset state of System object

Syntax

`reset(channel)`

Description

`reset(channel)` resets the internal state of the `phased.IsoSpeedUnderwaterPaths` object, `channel`. This method resets the coherence time clock.

Input Arguments

`channel` — Isospeed underwater channel path
`phased.IsoSpeedUnderwaterPaths` System object

Isospeed underwater channel paths, specified as a `phased.IsoSpeedUnderwaterPaths` System object.

Example: `phased.IsoSpeedUnderwaterPaths`

Version History

Introduced in R2017a

phased.IsotropicAntennaElement

Package: `phased`

Isotropic antenna element

Description

The `phased.IsotropicAntennaElement` object creates an antenna element with an isotropic response pattern. The object models an antenna element whose response is unity in all directions. An isotropic antenna does not support polarization.

To compute the response of the antenna element for specified directions:

1. Create the `phased.IsotropicAntennaElement` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
antenna = phased.IsotropicAntennaElement
antenna = phased.IsotropicAntennaElement(Name,Value)
```

Description

`antenna = phased.IsotropicAntennaElement` creates an isotropic antenna System object, `antenna`, with default property values.

`antenna = phased.IsotropicAntennaElement(Name,Value)` creates an isotropic antenna object, `antenna`, with each specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`FrequencyRange` — Operating frequency range
`[0 1e20]` (default) | nonnegative, real-valued 1-by-2 row vector

Operating frequency range of the antenna, specified as a nonnegative, real-valued, 1-by-2 row vector in the form `[LowerBound HigherBound]`. The antenna element has no response outside the specified frequency range. Units are in Hz.

Data Types: `double`

`BackBaffled` — Backbaffle antenna element
`false` (default) | `true`

Backbaffle the antenna element, specified as `false` or `true`. Set this property to `true` to baffle the response on the backside of the antenna element. In this case, the antenna response to all azimuth angles beyond ±90° from broadside (0° azimuth and 0°elevation) is zero. When the value of this property is `false`, the back of the antenna element is not baffled.

Data Types: `logical`

Usage

Syntax

`RESP = antenna(FREQ,ANG)`

Description

`RESP = antenna(FREQ,ANG)` returns the antenna voltage response `RESP` at operating frequencies specified in `FREQ` and in directions specified in `ANG`.

Input Arguments

`FREQ` — Operating frequency of antenna element
nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

`FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the response is returned as `–Inf`. Element objects use the `FrequencyRange` property, except for `phased.CustomAntennaElement`, which uses the `FrequencyVector` property.

Example: `[1e8 2e6]`

Data Types: `double`

`ANG` — Azimuth and elevation angles of response directions
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If `ANG` is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.

- If ANG is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of "Azimuth and Elevation Angles".

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

RESP — Voltage response of antenna
complex-valued *M*-by-*L* matrix

Voltage response of antenna element, returned as a complex-valued *M*-by-*L* matrix. In this matrix, *M* represents the number of angles specified in `ANG` and *L* represents the number of frequencies specified in `FREQ`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Plot Isotropic Antenna Element Response

Create an isotropic antenna operating over a frequency range from 800 MHz to 1.2 GHz. The operating frequency is 1 GHz. Find the response of the antenna at boresight. Then, plot the polar-pattern elevation response of the antenna.

```
antenna = phased.IsotropicAntennaElement( ...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
```

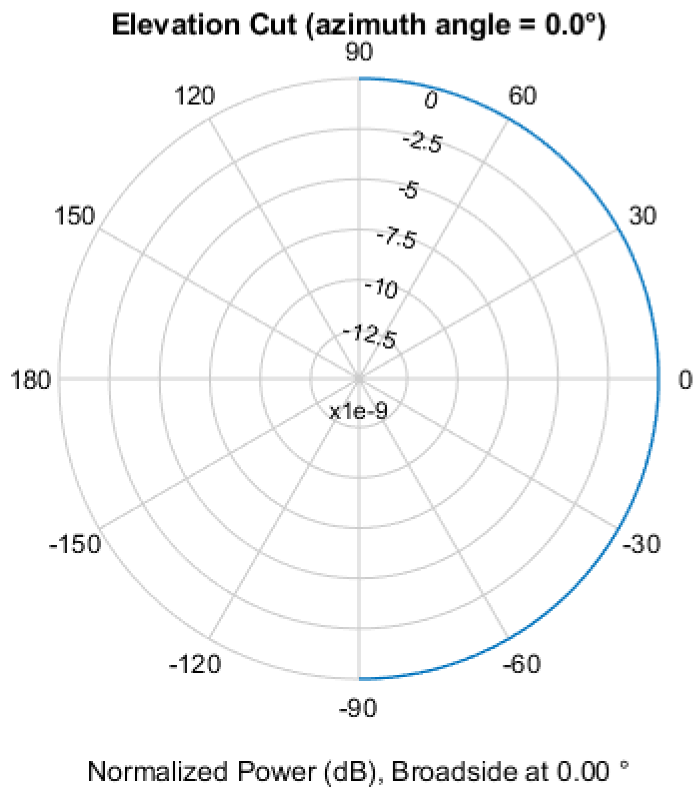
Obtain the response at boresight.

```
resp = antenna(fc,[0;0])
```

```
resp = 1
```

Plot the response pattern.

```
pattern(antenna,fc,0,[-90:90],'CoordinateSystem','polar', ...
    'Type','powerdb','Normalize',true)
```



Isotropic Antenna Does Not Support Polarization

Create an isotropic antenna element using the `phased.IsotropicAntennaElement` System object™ and show that it does not support polarization.

```
antenna = phased.IsotropicAntennaElement('FrequencyRange',[1.0,10]*1e9);
isPolarizationCapable(antenna)
```

```
ans = logical
   0
```

The returned value `0` shows that the antenna element does not support polarization.

Directivity of Isotropic Antenna Element

Compute the directivity of an isotropic antenna element in different directions.

Create an isotropic antenna element System object™.

```
antenna = phased.IsotropicAntennaElement();
```

First, specify that the directions of interest all at an elevation angle of zero degrees. The seven azimuth angles are centered around boresight (zero degrees azimuth and zero degrees elevation). Set the frequency to 1 GHz.

```
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
freq = 1e9;
```

Compute the directivity along the constant elevation cut.

```
d = directivity(antenna,freq,ang)
```

```
d = 7×1

     0
     0
     0
     0
     0
     0
     0
```

Next, specify that the directions of interest all at an azimuth angle of zero degrees. All elevation angles are centered around boresight. The five elevation angles range from −20 to +20 degrees, inclusive. Set the frequency to 1 GHz.

```
ang = [0,0,0,0,0; -20,-10,0,10,20];
freq = 1e9;
```

Compute the directivity along the constant azimuth cut.

```
d = directivity(antenna,freq,ang)
```

```
d = 5×1

     0
     0
     0
     0
     0
```

For an isotropic antenna, the directivity is independent of direction.

Plot Pattern and Directivity of Isotropic Antenna

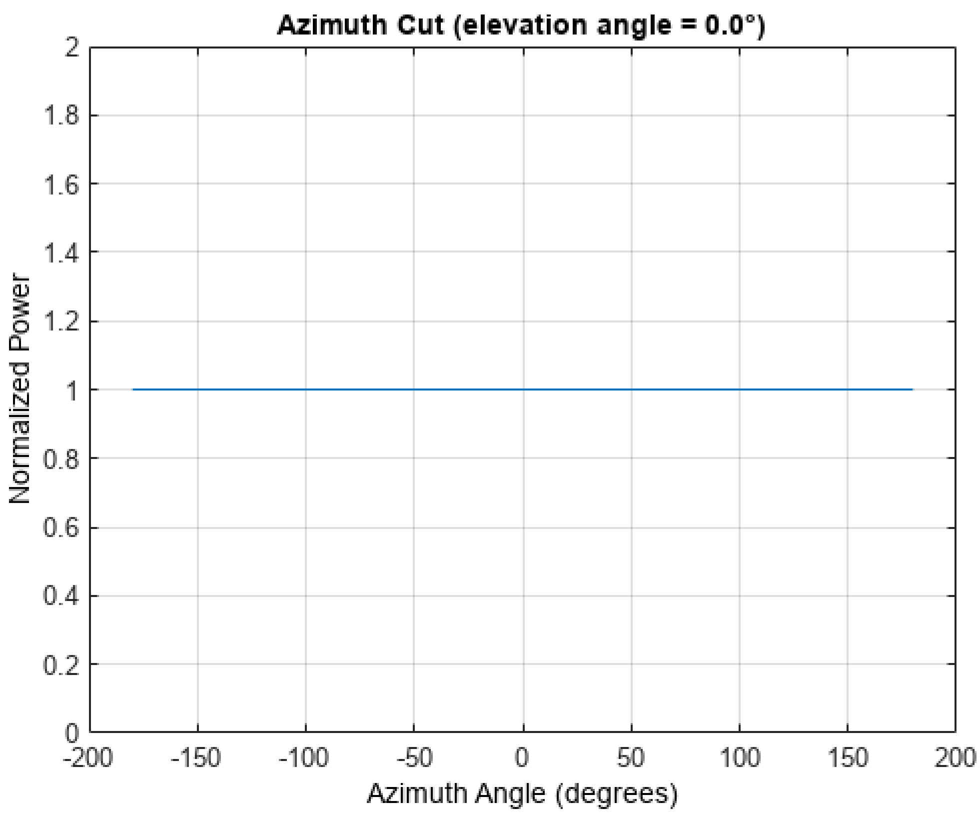
Create an isotropic antenna element. Then, plot the antenna power pattern and directivity.

First, create the antenna.

```
antenna = phased.IsotropicAntennaElement;
```

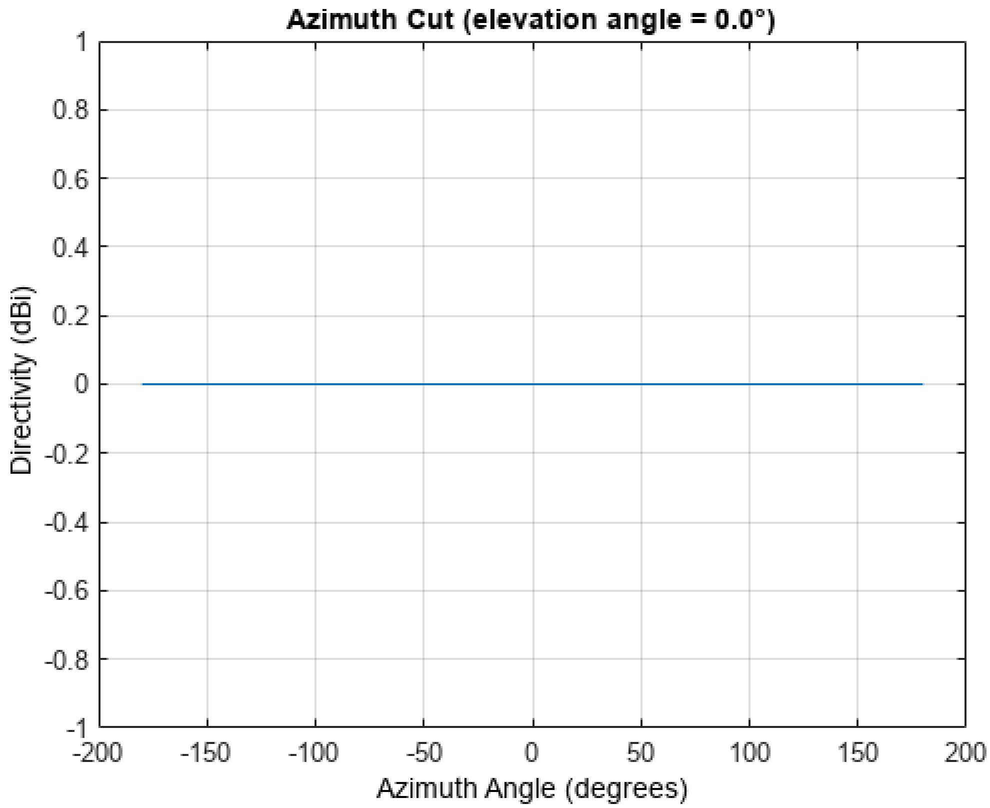
Draw an azimuth cut of the power pattern at 0 degrees elevation. Assume the operating frequency is 1 GHz.

```
fc = 1e9;
pattern(antenna,fc,[-180:180],0,...
    'Type','power',...
    'CoordinateSystem','rectangular')
```



Draw the same azimuth cut of the antenna directivity.

```
pattern(antenna,fc,[-180:180],0,...
    'Type','directivity',...
    'CoordinateSystem','rectangular')
```



Elevation-Cut of Isotropic Antenna Pattern

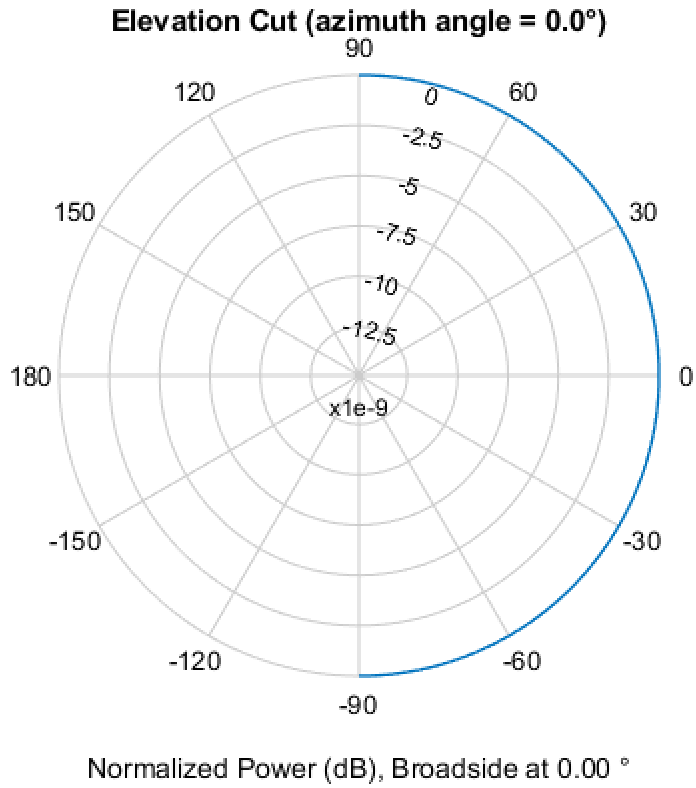
Construct an isotropic antenna operating in the frequency range from 800 MHz to 1.2 GHz. Compute the response at boresight at 1 GHz. Display the power pattern of the antenna at 1 GHz.

```
antenna = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
resp = antenna(fc,[0;0])
```

```
resp = 1
```

Plot the elevation power pattern of the antenna in polar coordinates.

```
pattern(antenna,fc,0,[-90:90],'Type','powerdb','CoordinateSystem','polar')
```



3-D Isotropic Antenna Pattern

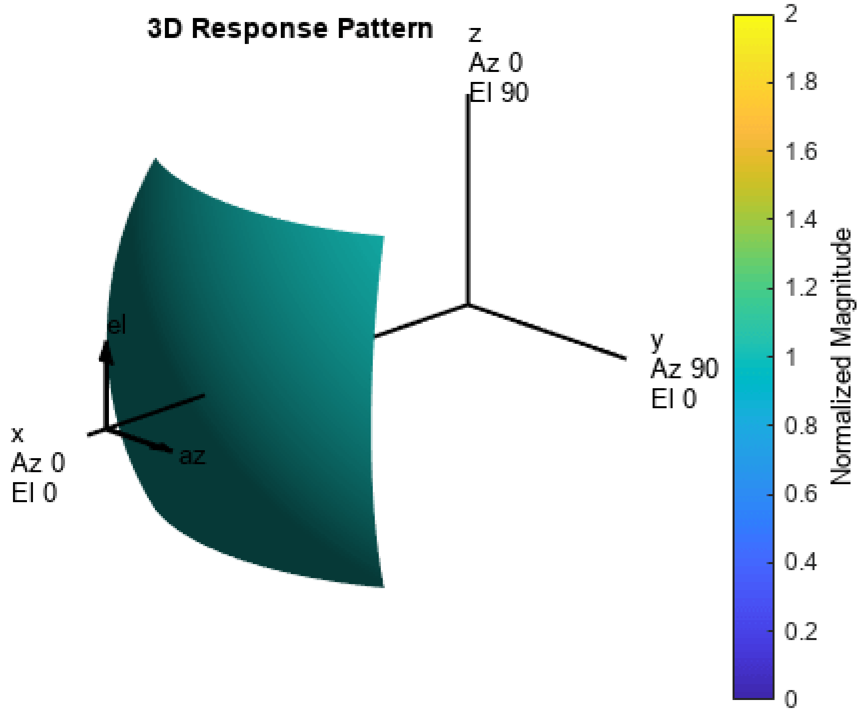
Construct an isotropic antenna operating over a frequency range from 800 MHz to 1.2 GHz. Then, plot the 3-D antenna field pattern.

Construct an isotropic antenna element.

```
antenna = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
```

Plot the 3-D magnitude pattern of the antenna at 1 GHz from −30 to 30 degrees in both azimuth and elevation in 0.1 degree increments.

```
fc = 1e9;
pattern(antenna,fc,[-30:0.1:30],[-30:0.1:30],...
    'Type','efield',...
    'CoordinateSystem','polar')
```



Directivity Pattern of Isotropic Antenna Element for Span of Azimuth Angles

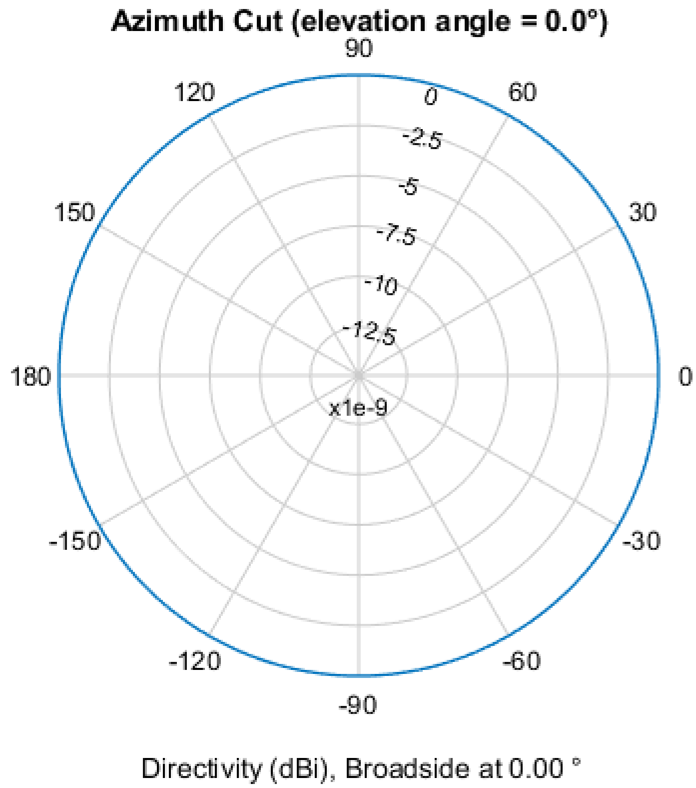
Plot an azimuth cut of the directivity of an isotropic antenna element at 0 degrees elevation for all azimuth angles and at 30 degrees elevation for a small span of azimuth angles. Assume the operating frequency is 500 MHz.

Create the antenna element.

```
fc = 500e6;
antenna = phased.IsotropicAntennaElement('FrequencyRange',[100,900]*1e6);
```

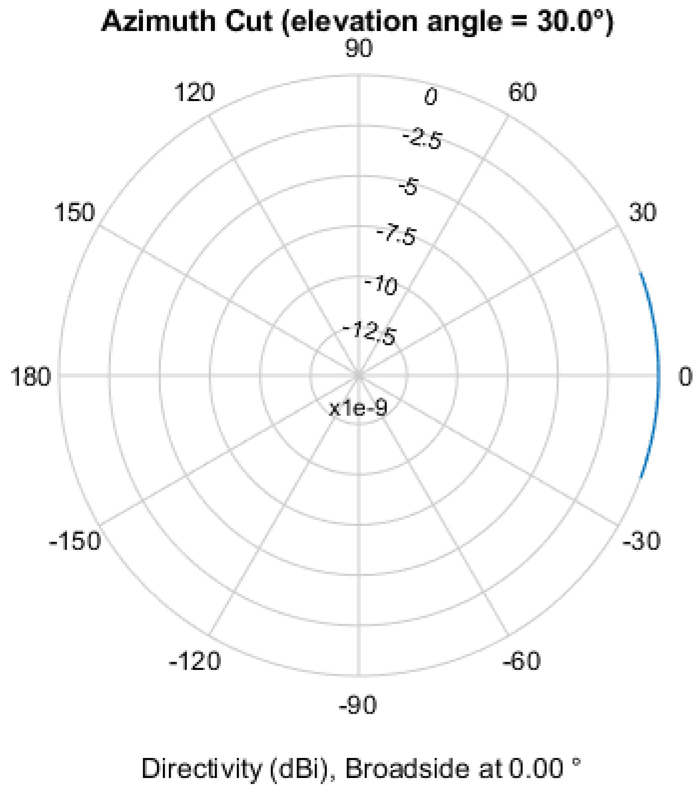
Plot the pattern for all azimuth angles at 0 degrees elevation.

```
patternAzimuth(antenna,fc,0)
```



Plot the pattern for a reduced span of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(antenna,fc,30,'Azimuth',[-20:20])
```



Directivity Pattern of Isotropic Antenna Element for Span of Elevation Angles

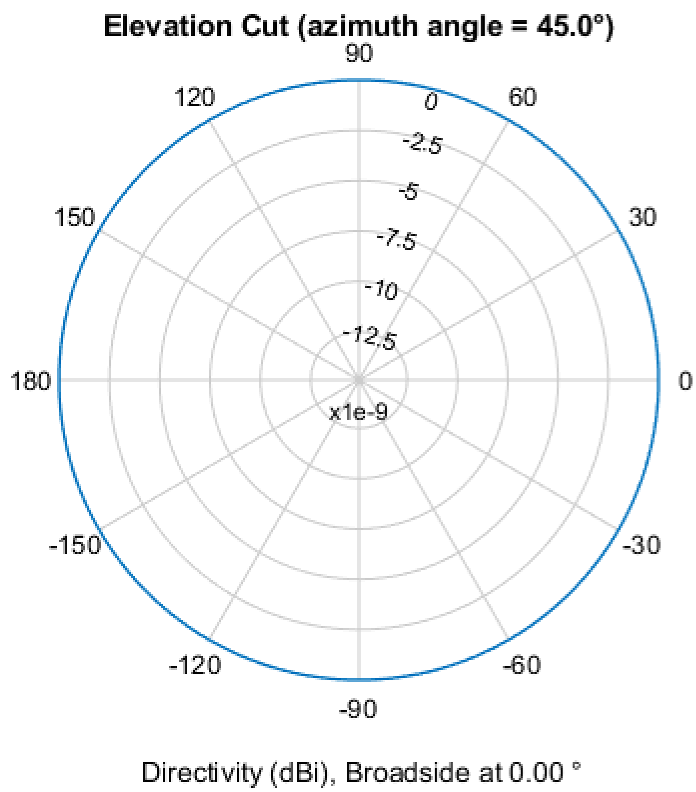
Plot an elevation cut of directivity of an isotropic antenna element at 45 degrees azimuth for all elevation angles and at 45 degrees for a span of elevation angles. Assume the operating frequency is 500 MHz.

Create the antenna element.

```
fc = 500e6;
antenna = phased.IsotropicAntennaElement('FrequencyRange',[100,900]*1e6);
```

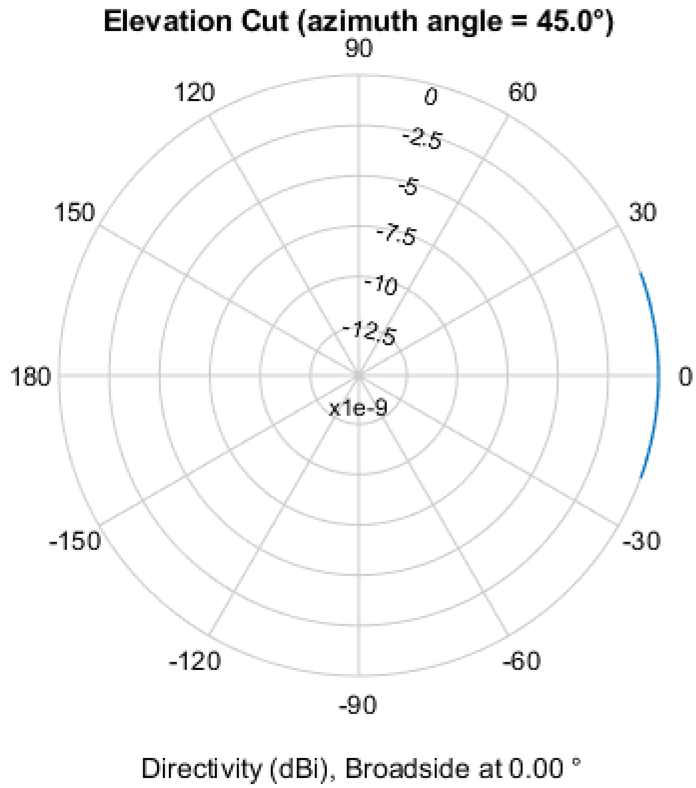
Plot the directivity for all elevation angles.

```
patternElevation(antenna,fc,45)
```



Plot the directivity for a span of elevation angles using the `Elevation` parameter.

```
patternElevation(antenna,fc,45,'Elevation',[-20:20])
```



Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, and `patternElevation` object functions are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CrossedDipoleAntennaElement` | `phased.CustomAntennaElement` | `phased.CustomMicrophoneElement` | `phased.OmnidirectionalMicrophoneElement` | `phased.ShortDipoleAntennaElement` | `phased.ULA` | `phased.URA` | `uv2azel` | `phitheta2azel`

directivity

System object: `phased.IsotropicAntennaElement`
Package: `phased`

Directivity of isotropic antenna element

Syntax

```
D = directivity(H,FREQ,ANGLE)
```

Description

`D = directivity(H,FREQ,ANGLE)` returns the “Directivity (dBi)” on page 1-763 of an isotropic antenna element, `H`, at frequencies specified by `FREQ` and in direction angles specified by `ANGLE`.

Input Arguments

`H` — Isotropic antenna element

System object

Isotropic antenna element specified as a `phased.IsotropicAntennaElement` System object.

Example: `H = phased.IsotropicAntennaElement;`

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`ANGLE` — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Output Arguments

D — Directivity

M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Isotropic Antenna Element

Compute the directivity of an isotropic antenna element in different directions.

Create an isotropic antenna element System object™.

```
antenna = phased.IsotropicAntennaElement();
```

First, specify that the directions of interest all at an elevation angle of zero degrees. The seven azimuth angles are centered around boresight (zero degrees azimuth and zero degrees elevation). Set the frequency to 1 GHz.

```
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
freq = 1e9;
```

Compute the directivity along the constant elevation cut.

```
d = directivity(antenna,freq,ang)
```

```
d = 7×1

     0
     0
     0
     0
     0
     0
     0
```

Next, specify that the directions of interest all at an azimuth angle of zero degrees. All elevation angles are centered around boresight. The five elevation angles range from −20 to +20 degrees, inclusive. Set the frequency to 1 GHz.

```
ang = [0,0,0,0,0; -20,-10,0,10,20];
freq = 1e9;
```

Compute the directivity along the constant azimuth cut.

```
d = directivity(antenna,freq,ang)
```

```
d = 5×1

     0
     0
     0
     0
     0
```

For an isotropic antenna, the directivity is independent of direction.

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`patternAzimuth` | `pattern` | `patternElevation`

isPolarizationCapable

System object: phased.IsotropicAntennaElement
Package: phased

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.IsotropicAntennaElement` System object supports polarization. An antenna element supports polarization if it can create or respond to polarized fields. This object does not support polarization.

Input Arguments

h — Isotropic antenna element

Isotropic antenna element specified as a `phased.IsotropicAntennaElement` System object.

Output Arguments

flag — Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not. Since the `phased.IsotropicAntennaElement` object does not support polarization, `flag` is always returned as `false`.

Examples

Isotropic Antenna Does Not Support Polarization

Create an isotropic antenna element using the `phased.IsotropicAntennaElement` System object™ and show that it does not support polarization.

```
antenna = phased.IsotropicAntennaElement('FrequencyRange',[1.0,10]*1e9);
isPolarizationCapable(antenna)
```

```
ans = logical
   0
```

The returned value `0` shows that the antenna element does not support polarization.

pattern

System object: `phased.IsotropicAntennaElement`
Package: `phased`

Plot isotropic antenna element directivity and patterns

Syntax

```
pattern(sElem,FREQ)
pattern(sElem,FREQ,AZ)
pattern(sElem,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sElem,FREQ)` plots the 3-D array directivity pattern (in dBi) for the element specified in `sElem`. The operating frequency is specified in `FREQ`.

`pattern(sElem,FREQ,AZ)` plots the element directivity pattern at the specified azimuth angle.

`pattern(sElem,FREQ,AZ,EL)` plots the element directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the element pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See "Convert plotResponse to pattern" on page 1-772 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sElem` — Isotropic antenna element
System object

Isotropic antenna element, specified as a `phased.IsotropicAntennaElement` System object.

Example: `sElem = phased.IsotropicAntennaElement;`

`FREQ` — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the

pattern azimuth and elevation, respectively. AZ values must lie between –180° and 180°. EL values must lie between –90° and 90°. If 'CoordinateSystem' is set to 'uv', AZ and EL then specify *U* and *V* coordinates, respectively. AZ and EL must lie between -1 and 1.

Example: 'uv'

Data Types: char

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

PlotStyle — Plotting style
'overlay' (default) | 'waterfall'

Plotting style, specified as the comma-separated pair consisting of 'Plotstyle' and either 'overlay' or 'waterfall'. This parameter applies when you specify multiple frequencies in FREQ in 2-D plots. You can draw 2-D plots by setting one of the arguments AZ or EL to a scalar.

Data Types: char

Output Arguments

PAT — Element pattern
N-by-*M* real-valued matrix

Element pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of PAT correspond to the azimuth angles in the vector specified by EL_ANG. The columns correspond to the elevation angles in the vector specified by AZ_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

Examples

Plot Pattern and Directivity of Isotropic Antenna

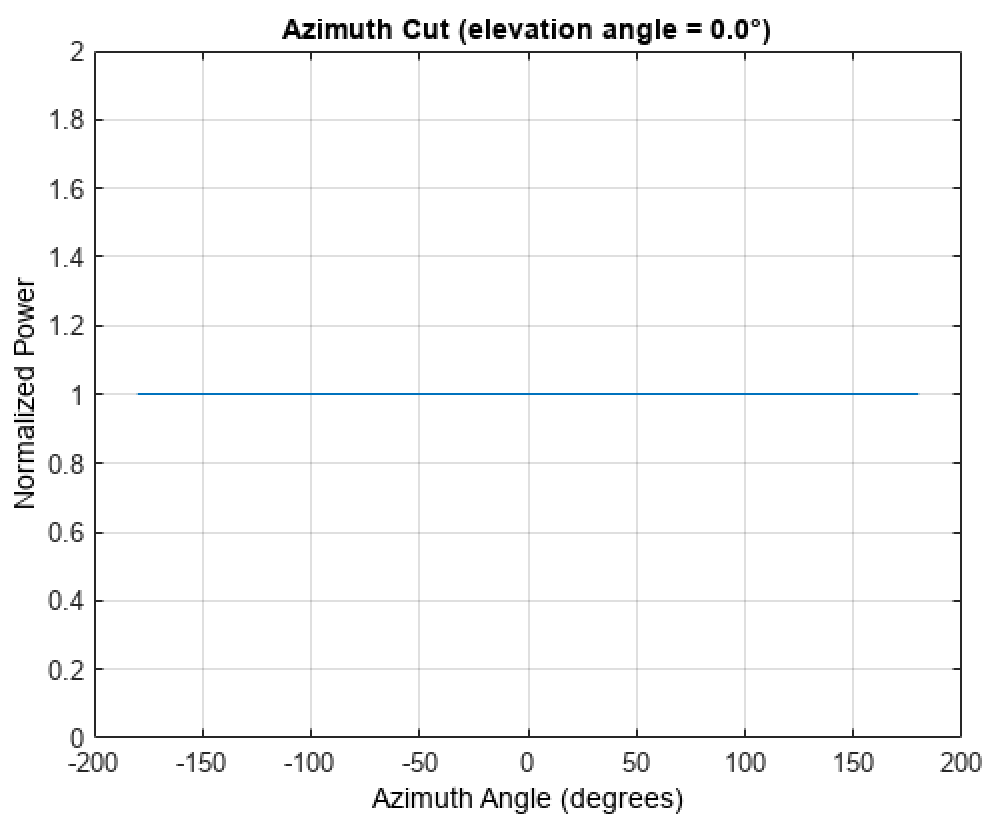
Create an isotropic antenna element. Then, plot the antenna power pattern and directivity.

First, create the antenna.

```
antenna = phased.IsotropicAntennaElement;
```

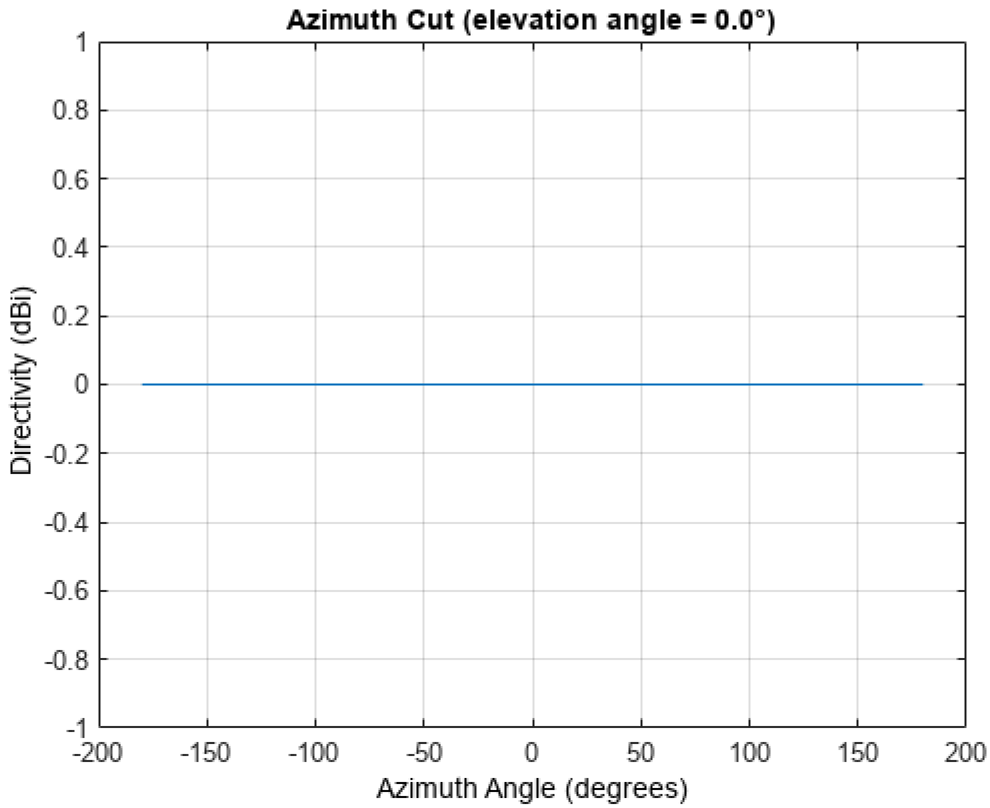
Draw an azimuth cut of the power pattern at 0 degrees elevation. Assume the operating frequency is 1 GHz.

```
fc = 1e9;
pattern(antenna,fc,[-180:180],0,...
    'Type','power',...
    'CoordinateSystem','rectangular')
```



Draw the same azimuth cut of the antenna directivity.

```
pattern(antenna,fc,[-180:180],0,...
    'Type','directivity',...
    'CoordinateSystem','rectangular')
```



Elevation-Cut of Isotropic Antenna Pattern

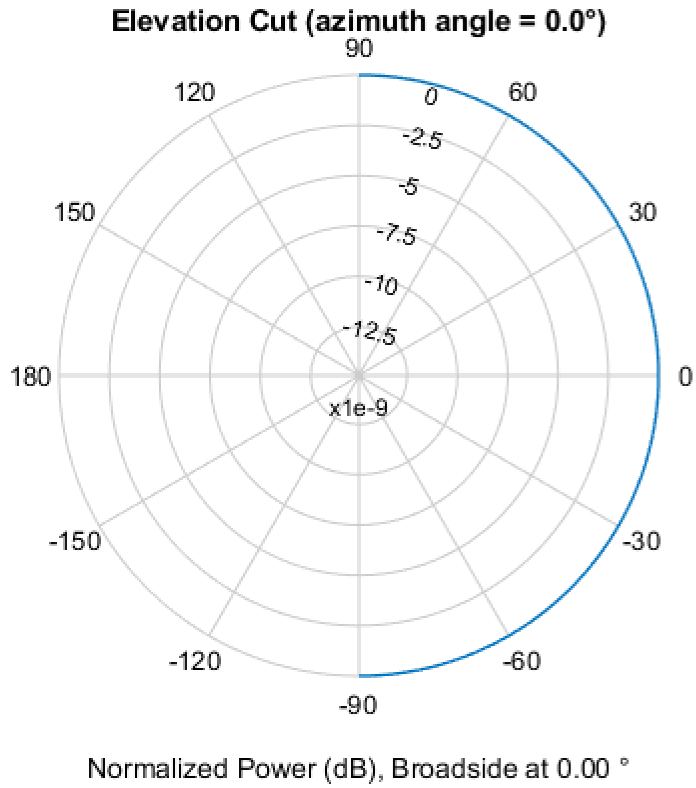
Construct an isotropic antenna operating in the frequency range from 800 MHz to 1.2 GHz. Compute the response at boresight at 1 GHz. Display the power pattern of the antenna at 1 GHz.

```
antenna = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
resp = antenna(fc,[0;0])
```

```
resp = 1
```

Plot the elevation power pattern of the antenna in polar coordinates.

```
pattern(antenna,fc,0,[-90:90],'Type','powerdb','CoordinateSystem','polar')
```



3-D Isotropic Antenna Pattern

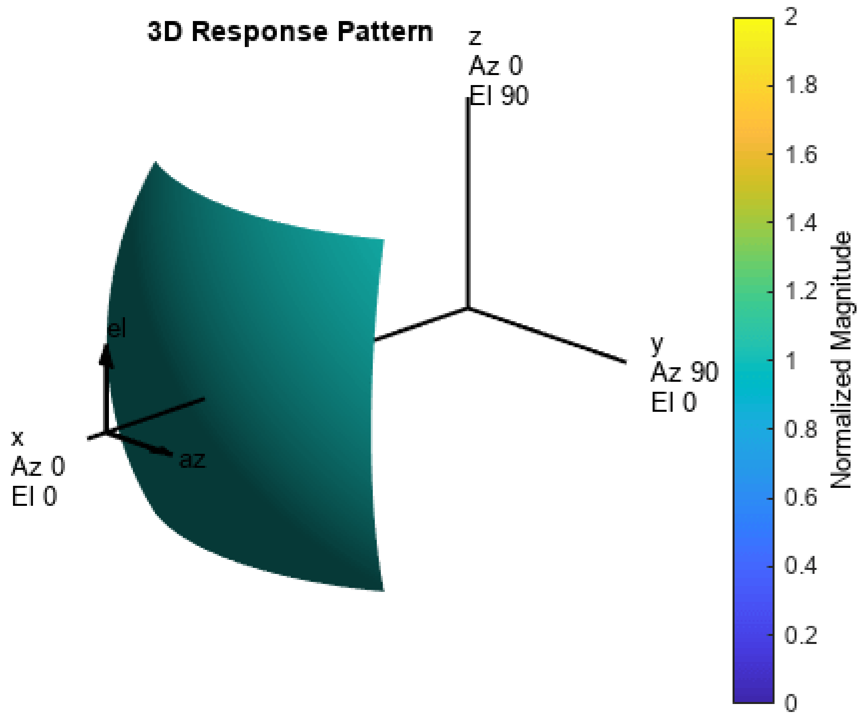
Construct an isotropic antenna operating over a frequency range from 800 MHz to 1.2 GHz. Then, plot the 3-D antenna field pattern.

Construct an isotropic antenna element.

```
antenna = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
```

Plot the 3-D magnitude pattern of the antenna at 1 GHz from −30 to 30 degrees in both azimuth and elevation in 0.1 degree increments.

```
fc = 1e9;
pattern(antenna,fc,[-30:0.1:30],[-30:0.1:30],...
    'Type','efield',...
    'CoordinateSystem','polar')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td><code>'Format'</code> and <code>'RespCut'</code> name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using <code>plotResponse</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'RespCut'</code> to <code>'Az' or 'El'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using either the <code>'AzimuthAngles'</code> or <code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'RespCut'</code> to <code>'3D'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using both the <code>'AzimuthAngles'</code> and<code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'RespCut'</code> to<code>'U'</code>. Set <code>'Format'</code> to <code>'UV'</code>. Set the display range using the <code>'UGrid'</code> name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'RespCut'</code> to<code>'3D'</code>. Set <code>'Format'</code> to</td></tr>
</table>
</td>
<td><code>'CoordinateSystem'</code> name-value pair used together with the <code>AZ</code> and <code>EL</code> input arguments.

<code>'CoordinateSystem'</code> has the same options as the <code>plotResponse</code> method <code>'Format'</code>name-value pair, except that <code>'line'</code> is now named <code>'rectangular'</code>. The table shows how to create different types of plots using <code>pattern</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify either <code>AZ</code> or <code>EL</code> as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify both <code>AZ</code> and <code>EL</code> as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space vector.</td></tr>
</table>
If you set <code>CoordinateSystem</code> to <code>'uv'</code>, enter the UV grid values using <code>AZ</code> and <code>EL</code>.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `patternElevation`

patternAzimuth

System object: phased.IsotropicAntennaElement
Package: phased

Plot isotropic antenna element directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sElem,FREQ)
patternAzimuth(sElem,FREQ,EL)
patternAzimuth(sElem,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sElem,FREQ)` plots the 2-D element directivity pattern versus azimuth (in dBi) for the element `sElem` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

`patternAzimuth(sElem,FREQ,EL)`, in addition, plots the 2-D element directivity pattern versus azimuth (in dBi) at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sElem,FREQ,EL,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sElem` — Isotropic antenna element

System object

Isotropic antenna element, specified as a `phased.IsotropicAntennaElement` System object.

Example: `sElem = phased.IsotropicAntennaElement;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

Examples

Directivity Pattern of Isotropic Antenna Element for Span of Azimuth Angles

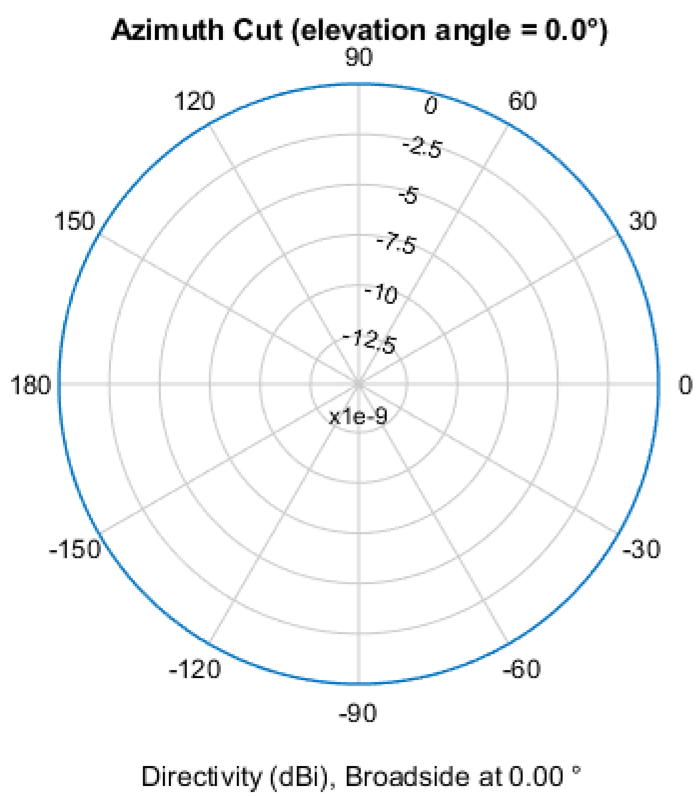
Plot an azimuth cut of the directivity of an isotropic antenna element at 0 degrees elevation for all azimuth angles and at 30 degrees elevation for a small span of azimuth angles. Assume the operating frequency is 500 MHz.

Create the antenna element.

```
fc = 500e6;
antenna = phased.IsotropicAntennaElement('FrequencyRange',[100,900]*1e6);
```

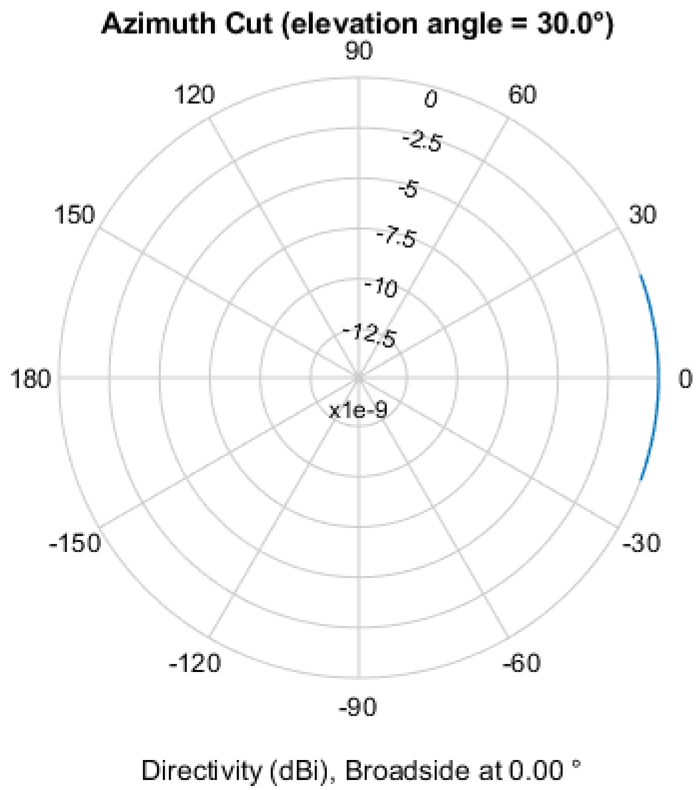
Plot the pattern for all azimuth angles at 0 degrees elevation.

```
patternAzimuth(antenna,fc,0)
```



Plot the pattern for a reduced span of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(antenna,fc,30,'Azimuth',[-20:20])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: phased.IsotropicAntennaElement
Package: phased

Plot isotropic antenna element directivity or pattern versus elevation

Syntax

```
patternElevation(sElem,FREQ)
patternElevation(sElem,FREQ,AZ)
patternElevation(sElem,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

patternElevation(sElem,FREQ) plots the 2-D element directivity pattern versus elevation (in dBi) for the element sElem at zero degrees azimuth angle. The argument FREQ specifies the operating frequency.

patternElevation(sElem,FREQ,AZ), in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by AZ. When AZ is a vector, multiple overlaid plots are created.

patternElevation(sElem,FREQ,AZ,Name,Value) plots the element pattern with additional options specified by one or more Name,Value pair arguments.

PAT = patternElevation(___) returns the element pattern. PAT is a matrix whose entries represent the pattern at corresponding sampling points specified by the 'Elevation' parameter and the AZ input argument.

Input Arguments

sElem — Isotropic antenna element
System object

Isotropic antenna element, specified as a phased.IsotropicAntennaElement System object.

Example: sElem = phased.IsotropicAntennaElement;

FREQ — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

Elevation — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Directivity Pattern of Isotropic Antenna Element for Span of Elevation Angles

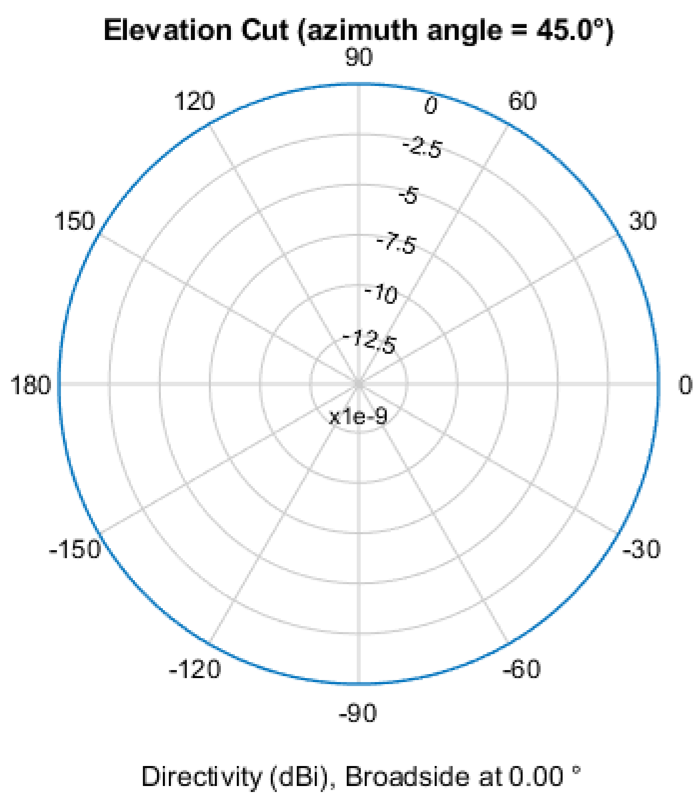
Plot an elevation cut of directivity of an isotropic antenna element at 45 degrees azimuth for all elevation angles and at 45 degrees for a span of elevation angles. Assume the operating frequency is 500 MHz.

Create the antenna element.

```
fc = 500e6;
antenna = phased.IsotropicAntennaElement('FrequencyRange',[100,900]*1e6);
```

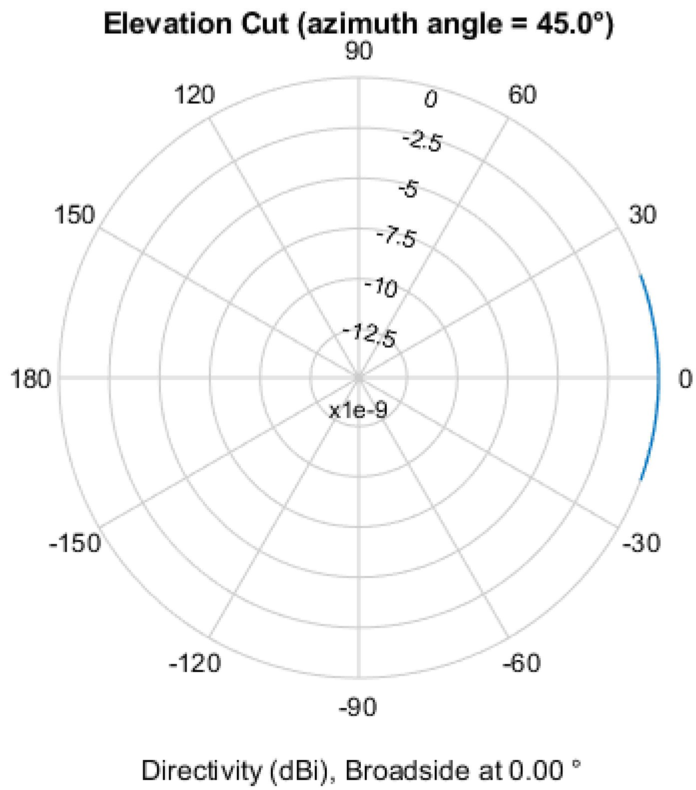
Plot the directivity for all elevation angles.

```
patternElevation(antenna,fc,45)
```



Plot the directivity for a span of elevation angles using the `Elevation` parameter.

```
patternElevation(antenna,fc,45,'Elevation',[-20:20])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `pattern`

plotResponse

System object: `phased.IsotropicAntennaElement`
Package: `phased`

Plot response pattern of antenna

Syntax

```
plotResponse(H,FREQ)
plotResponse(H,FREQ,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ)` plots the element response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`.

`plotResponse(H,FREQ,Name,Value)` plots the element response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

`H`

Element System object

`FREQ`

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CutAngle`

Cut angle specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

Format

Format of the plot, using one of 'Line', 'Polar', or 'UV'. If you set Format to 'UV', FREQ must be a scalar.

Default: 'Line'

NormalizeResponse

Set this value to true to normalize the response pattern. Set this value to false to plot the response pattern without normalizing it. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: true

OverlayFreq

Set this value to true to overlay pattern cuts in a 2-D line plot. Set this value to false to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is false, FREQ must be a vector with at least two entries.

This parameter applies only when Format is not 'Polar' and RespCut is not '3D'.

Default: true

Polarization

Specify the polarization options for plotting the antenna response pattern. The allowable values are |'None' | 'Combined' | 'H' | 'V' | where

- 'None' specifies plotting a nonpolarized response pattern
- 'Combined' specifies plotting a combined polarization response pattern
- 'H' specifies plotting the horizontal polarization response pattern
- 'V' specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is 'None'. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: 'None'

RespCut

Cut of the response. Valid values depend on Format, as follows:

- If Format is 'Line' or 'Polar', the valid values of RespCut are 'Az', 'El', and '3D'. The default is 'Az'.
- If Format is 'UV', the valid values of RespCut are 'U' and '3D'. The default is 'U'.

If you set RespCut to '3D', FREQ must be a scalar.

Unit

The unit of the plot. Valid values are 'db', 'mag', 'pow', or 'dbi'. This parameter determines the type of plot that is produced.

Unit value	Plot type
db	power pattern in dB scale
mag	field pattern
pow	power pattern
dbi	directivity

Default: 'db'

AzimuthAngles

Azimuth angles for plotting element response, specified as a row vector. The AzimuthAngles parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'Az' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the AzimuthAngles and ElevationAngles parameters simultaneously.

Default: [-180:180]

ElevationAngles

Elevation angles for plotting element response, specified as a row vector. The ElevationAngles parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'El' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the ElevationAngles and AzimuthAngles parameters simultaneously.

Default: [-90:90]

UGrid

U coordinate values for plotting element response, specified as a row vector. The UGrid parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to 'U' or '3D'. The values of UGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the UGrid and VGrid parameters simultaneously.

Default: [-1:0.01:1]

VGrid

V coordinate values for plotting element response, specified as a row vector. The VGrid parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to '3D'. The values of VGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the VGrid and UGrid parameters simultaneously.

Default: [-1:0.01:1]

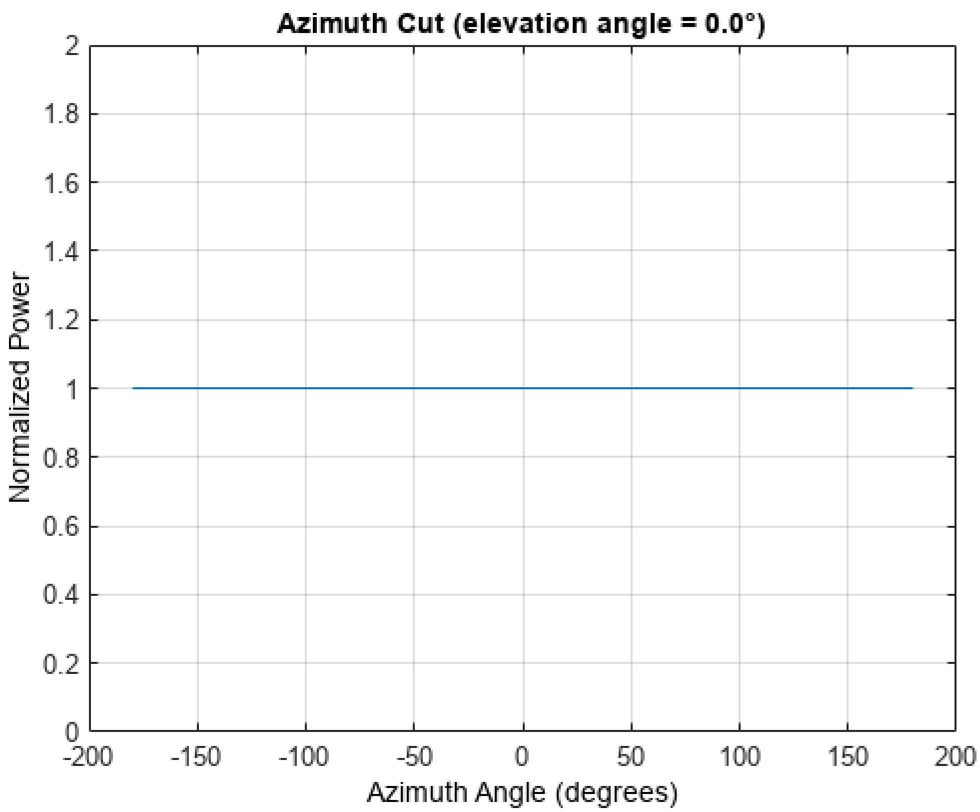
Examples

Plot Response and Directivity of Isotropic Antenna

This example shows how to plot the response and the directivity of an isotropic antenna element.

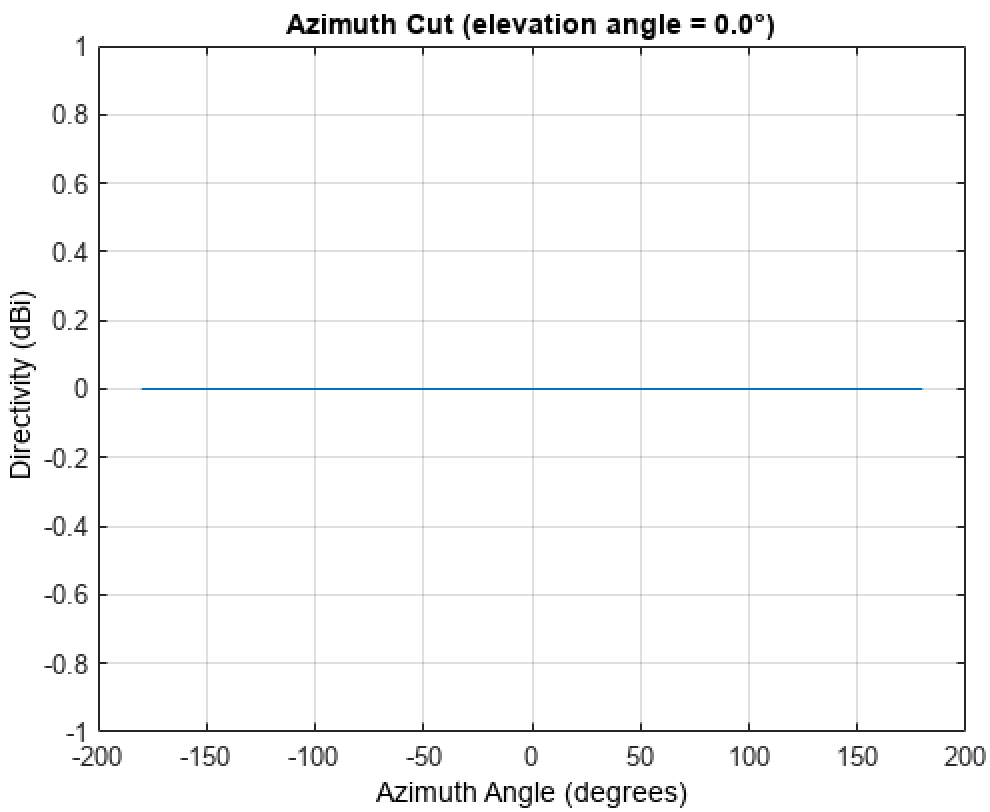
Draw a line plot of an azimuth cut of the response of an isotropic antenna along 0 degrees elevation. Assume the operating frequency is 1 GHz.

```
sIso = phased.IsotropicAntennaElement;
plotResponse(sIso,1e9,'Unit','pow');
```



Draw an azimuth cut of the antenna directivity.

```
plotResponse(sIso,1e9,'Unit','dbi');
```



Plot Elevation-Cut of Isotropic Antenna Response

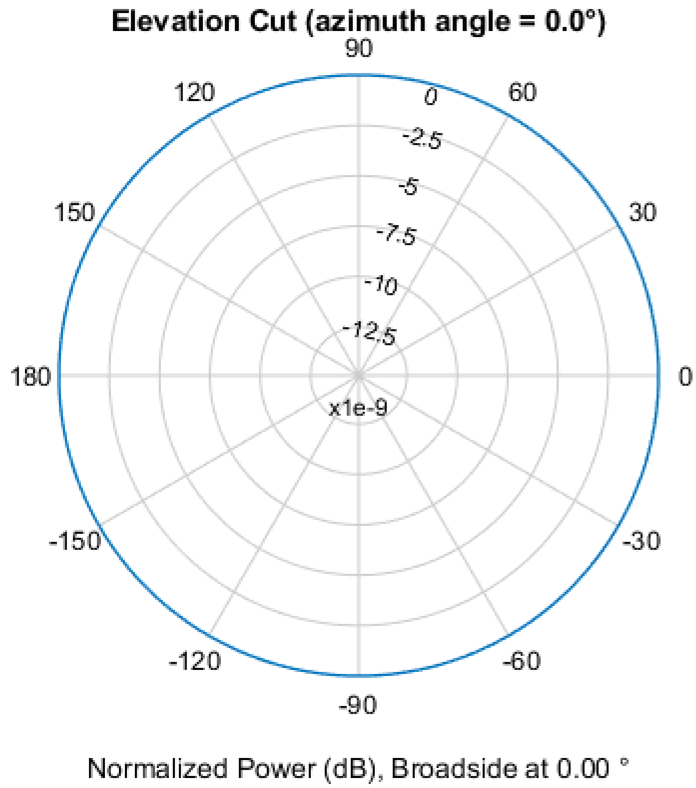
Construct an isotropic antenna operating in the frequency range from 800 MHz to 1.2 GHz. Find the response of the antenna at boresight at 1 GHz.

```
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
resp = step(sIso,fc,[0;0])
```

```
resp = 1
```

Plot the polar-form of the elevation response of the antenna.

```
plotResponse(sIso,fc,'RespCut','El','Format','Polar');
```



Plot 3-D Response

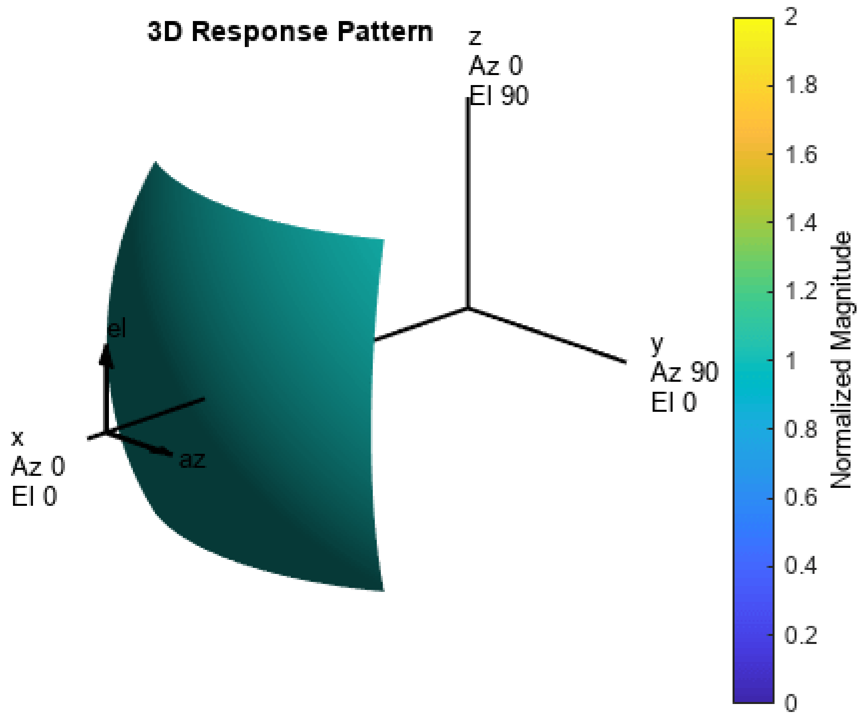
This example shows how to construct an isotropic antenna operating over a frequency range from 800 MHz to 1.2 GHz and how to plot its response.

Construct the antenna element.

```
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[0.8e9 1.2e9]);
```

Plot the 3-D response of the antenna at 1 GHz from -30 to 30 degrees in both azimuth and elevation at 0.1 degree increments.

```
fc = 1e9;
plotResponse(sIso,fc,'RespCut','3D','Format','Polar',...
    'Unit','mag','AzimuthAngles',[-30:.1:30],...
    'ElevationAngles',[-30:.1:30]);
```



See Also

`uv2azel` | `azel2uv`

step

System object: `phased.IsotropicAntennaElement`
Package: `phased`

Output response of antenna element

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the antenna's voltage response `RESP` at operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`H`

Antenna element object.

`FREQ`

Operating frequencies of antenna in hertz. `FREQ` is a row vector of length L.

`ANG`

Directions in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If `ANG` is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

Output Arguments

RESP

Voltage response of antenna element specified as an *M*-by-*L*, complex-valued matrix. In this matrix, *M* represents the number of angles specified in ANG while *L* represents the number of frequencies specified in FREQ.

Examples

Plot Isotropic Antenna Element Response

Create an isotropic antenna operating over a frequency range from 800 MHz to 1.2 GHz. The operating frequency is 1 GHz. Find the response of the antenna at boresight. Then, plot the polar-pattern elevation response of the antenna.

```
antenna = phased.IsotropicAntennaElement( ...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
```

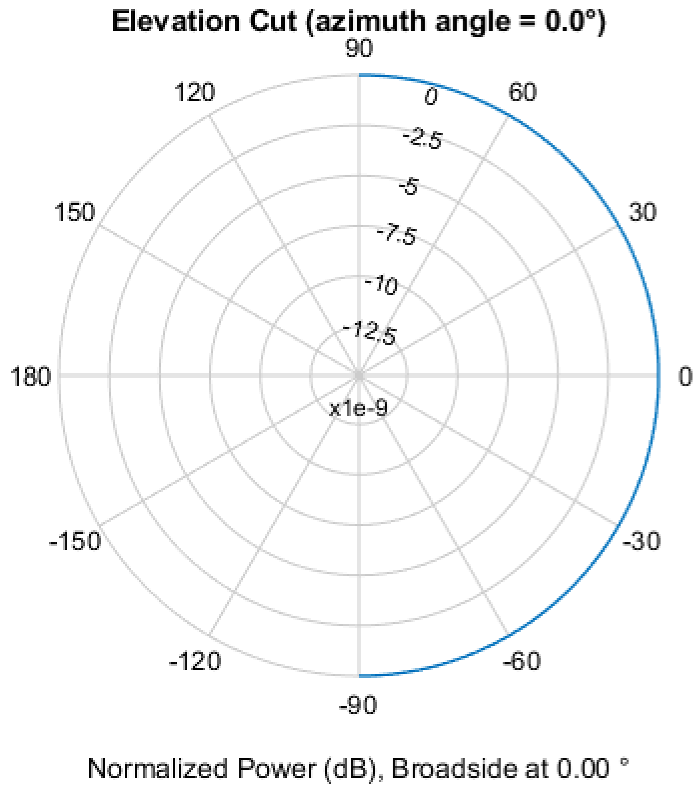
Obtain the response at boresight.

```
resp = antenna(fc,[0;0])
```

```
resp = 1
```

Plot the response pattern.

```
pattern(antenna,fc,0,[-90:90],'CoordinateSystem','polar', ...
    'Type','powerdb','Normalize',true)
```



See Also

`uv2azel` | `phitheta2azel`

phased.IsotropicHydrophone

Package: `phased`

Isotropic hydrophone element

Description

The `phased.IsotropicHydrophone` System object models an isotropic hydrophone for sonar applications. An isotropic hydrophone has the same response in all signal directions. The response is the output voltage of the hydrophone per unit sound pressure. The response of a hydrophone is also called its sensitivity. You can specify the response using the `VoltageSensitivity` property.

To compute the response of the isotropic hydrophone element for specified directions:

1. Create the `phased.IsotropicHydrophone` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
hydrophone = phased.IsotropicHydrophone
hydrophone = phased.IsotropicHydrophone(Name=Value)
```

Description

`hydrophone = phased.IsotropicHydrophone` creates an isotropic hydrophone System object, `hydrophone`.

`hydrophone = phased.IsotropicHydrophone(Name=Value)` creates an Isotropic hydrophone object, `hydrophone`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1=Value1`,...,`NameN=ValueN`).

Example: `hydrophone = phased.IsotropicHydrophone(FrequencyRange=[0 1000],BackBaffled=true)` creates an isotropic hydrophone element with its frequency range specified between 0 and 1000 Hz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`FrequencyRange` — Operating frequency range of hydrophone
`[0 100e6]` (default) | real-valued 1-by-2 vector

Operating frequency range of hydrophone, specified as a real-valued 1-by-2 row vector of the form `[LowerBound HigherBound]`. This property defines the frequency range over which the hydrophone has a response. The hydrophone element has zero response outside this frequency range. Units are in Hz.

Example: `[0 1000]`

Data Types: `double`

`VoltageSensitivity` — Voltage sensitivity of hydrophone
`-120` (default) | scalar | real-valued 1-by-*K* row vector

Voltage sensitivity of hydrophone, specified as a scalar or real-valued 1-by-*K* row vector. When you specify the voltage sensitivity as a scalar, that value applies to the entire frequency range specified by `FrequencyRange`. When you specify the voltage sensitivity as a vector, the frequency range is divided into K-1 equal intervals. The sensitivity values are assigned to the interval end points. The `step` method interpolates the voltage sensitivity for any frequency inside the frequency range. Units are in dB//1V/μPa. See “Hydrophone Sensitivity” on page 1-806 for more details.

Example: `10`

Data Types: `double`

`BackBaffled` — Back baffle hydrophone element
`false` (default) | `true`

Baffle the back direction of hydrophone element, specified as `false` or `true`. When `true`, the hydrophone responses to all azimuth angles beyond ±90 degrees from broadside (zero degrees azimuth and elevation) are zero.

When the value of this property is `false`, the back direction of the microphone element is not baffled.

Usage

Syntax

```
resp = hydrophone(freq,ang)
```

Description

`resp = hydrophone(freq,ang)` returns the voltage sensitivity for the hydrophone at the specified operating frequencies and in the specified directions of arriving signals.

Input Arguments

`freq` — Voltage sensitivity frequencies
positive real scalar | real-valued 1-by-*L* vector of positive values

Voltage sensitivity frequencies of hydrophone, specified as a positive real scalar or a real-valued 1-by-*L* vector of positive values. Units are in Hz.

Data Types: `double`

ang — Direction of arriving signals
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Direction of arriving signals, specified as a real-valued 1-by-*M* row vector or 2-by-*M* matrix. When `ang` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

When `ang` is a 1-by-*M* row vector, each element specifies the azimuth angle of the arriving signal. In this case, the corresponding elevation angle is assumed to be zero.

Data Types: `double`

Output Arguments

resp — Voltage sensitivity of hydrophone
real-valued *M*-by-*L* matrix

Voltage sensitivity of hydrophone, returned as a real-valued *M*-by-*L* matrix. *M* represents the number of angles specified in `ang`, and *L* represents the number of frequencies specified in `freq`. Units are in V/Pa.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Response of Isotropic Hydrophone

Compute the response of an isotropic hydrophone operating at 2 kHz. The hydrophone has default property values.

Obtain the voltage sensitivity at five different elevation angles: -30°, -15°, 0°, 15° and 30°. All azimuth angles are at 0°. The sensitivities are computed at the signal frequency of 2 kHz.

```
hydrophone = phased.IsotropicHydrophone;
fc = 2e3;
resp = hydrophone(fc,[0 0 0 0 0;-30 -15 0 15 30]);
```

Response and Pattern of Isotropic Hydrophone for Single Frequency

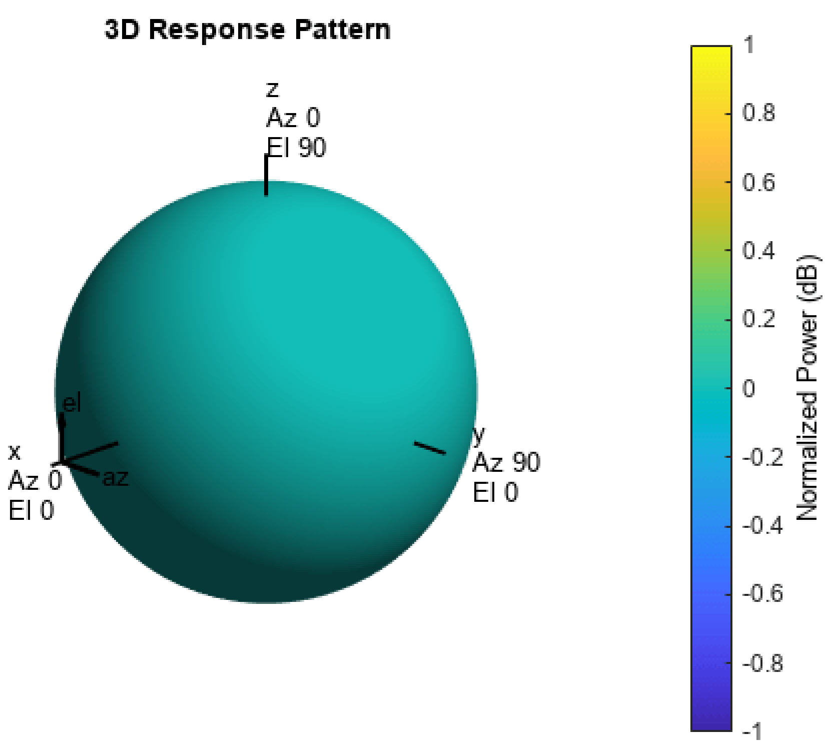
Examine the response and patterns of an isotropic hydrophone operating between 1 kHz and 10 kHz.

Set up the hydrophone parameters. Obtain the voltage sensitivity at five different elevation angles: -30°, -15°, 0°, 15° and 30°. All azimuth angles are at 0°. The sensitivities are computed at the signal frequency of 2 kHz.

```
hydrophone = phased.IsotropicHydrophone('FrequencyRange', ...
    [1 10]*1e3);
fc = 2e3;
resp = hydrophone(fc,[0 0 0 0 0;-30 -15 0 15 30]);
```

Draw a 3-D plot of the voltage sensitivity.

```
pattern(hydrophone,fc,[-180:180],[-90:90], ...
    'CoordinateSystem','polar','Type','powerdb')
```



Response and Pattern of Isotropic Hydrophone at Multiple Frequencies

Examine the response and patterns of an isotropic hydrophone at three different frequencies. The hydrophone operates between 1 kHz and 10 kHz. Specify the voltage sensitivity as a vector.

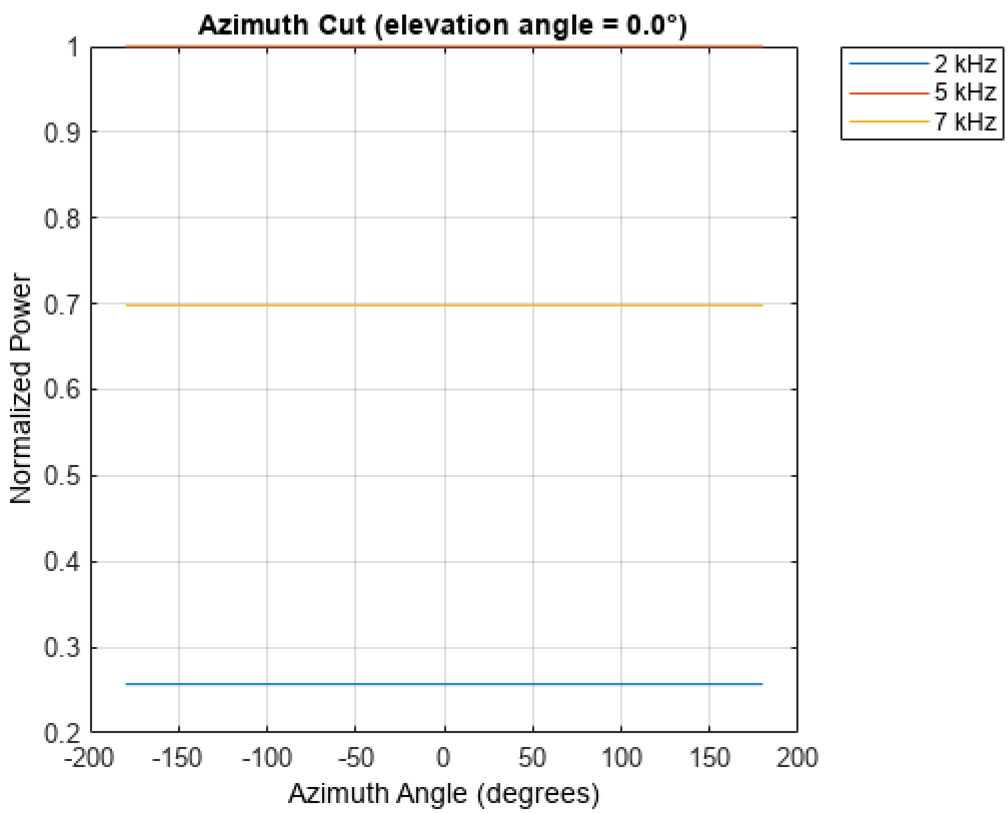
Set up the hydrophone parameters and obtain the voltage sensitivity at 45° azimuth and 30° elevation. Compute the sensitivities at the signal frequencies of 2, 5, and 7 kHz.

```
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1 10]*1e3, ...
    'VoltageSensitivity',[-100 -90 -100]);
fc = [2e3 5e3 7e3];
resp = hydrophone(fc,[45;30])
```

```
resp = 1×3

   14.8051   29.2202   24.4152
```

Draw a 2-D plot of the voltage sensitivity as a function of azimuth.

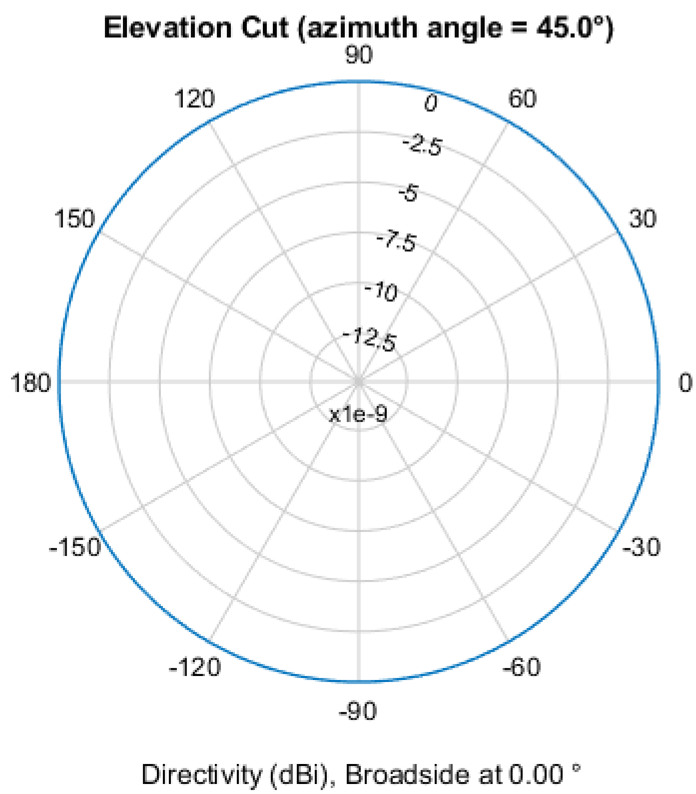
```
pattern(hydrophone,fc,[-180:180],0,'CoordinateSystem','rectangular',...
    'Type','power')
```



Directivity of Isotropic Hydrophone

Compute the directivity of an isotropic hydrophone in different directions. Assume the signal frequency is 3 kHz. First, set up the hydrophone parameters.

```
fc = 3e3;
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1,10]*1e3, ...
    'VoltageSensitivity',[-100,-90,-100]);
patternElevation(hydrophone,fc,45)
```



First, select the angles of interest to be constant elevation angle at zero degrees. The five azimuth angles are centered around boresight (zero degrees azimuth and zero degrees elevation).

```
ang = [-20,-10,0,10,20; 0,0,0,0,0];
```

Compute the directivity along the constant elevation cut.

```
d = directivity(hydrophone,fc,ang)
```

```
d = 5×1

     0
     0
     0
     0
     0
```

The directivity of an isotropic hydrophone is zero in every direction.

Response and Pattern of Isotropic Hydrophone at Multiple Frequencies

Examine the response and patterns of an isotropic hydrophone at three different frequencies. The hydrophone operates between 1 kHz and 10 kHz. Specify the voltage sensitivity as a vector.

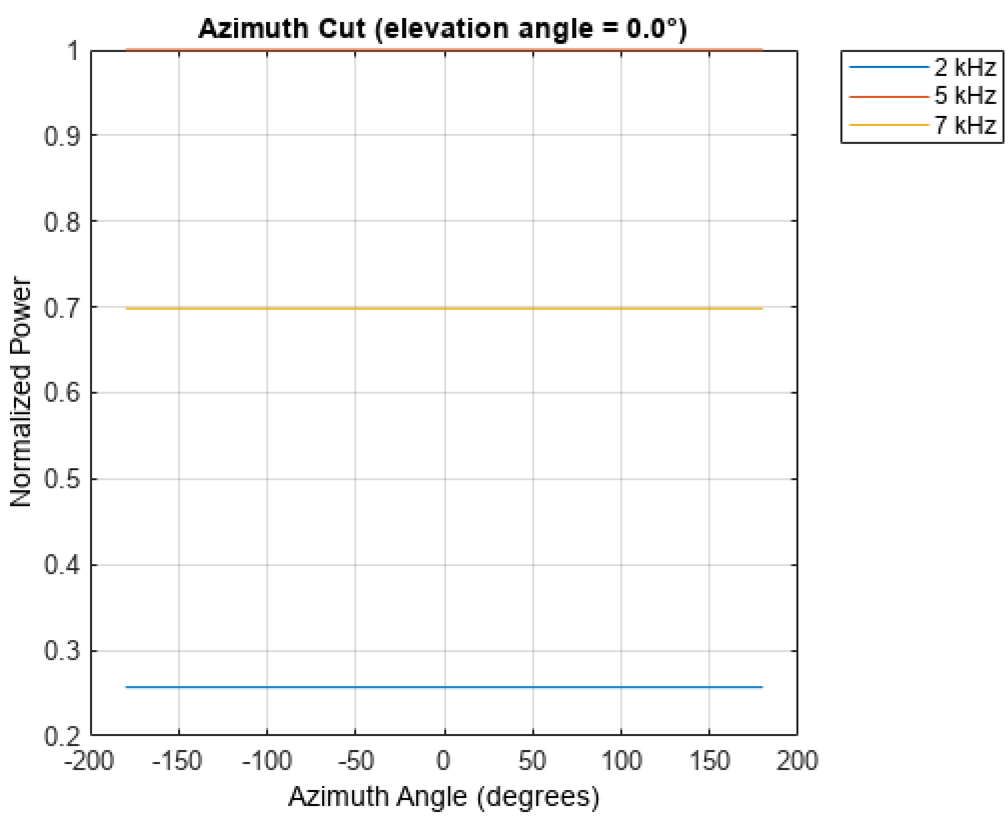
Set up the hydrophone parameters and obtain the voltage sensitivity at 45° azimuth and 30° elevation. Compute the sensitivities at the signal frequencies of 2, 5, and 7 kHz.

```
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1 10]*1e3, ...
    'VoltageSensitivity',[-100 -90 -100]);
fc = [2e3 5e3 7e3];
resp = hydrophone(fc,[45;30])
```

```
resp = 1×3

   14.8051   29.2202   24.4152
```

Draw a 2-D plot of the voltage sensitivity as a function of azimuth.

```
pattern(hydrophone,fc,[-180:180],0,'CoordinateSystem','rectangular',...
    'Type','power')
```

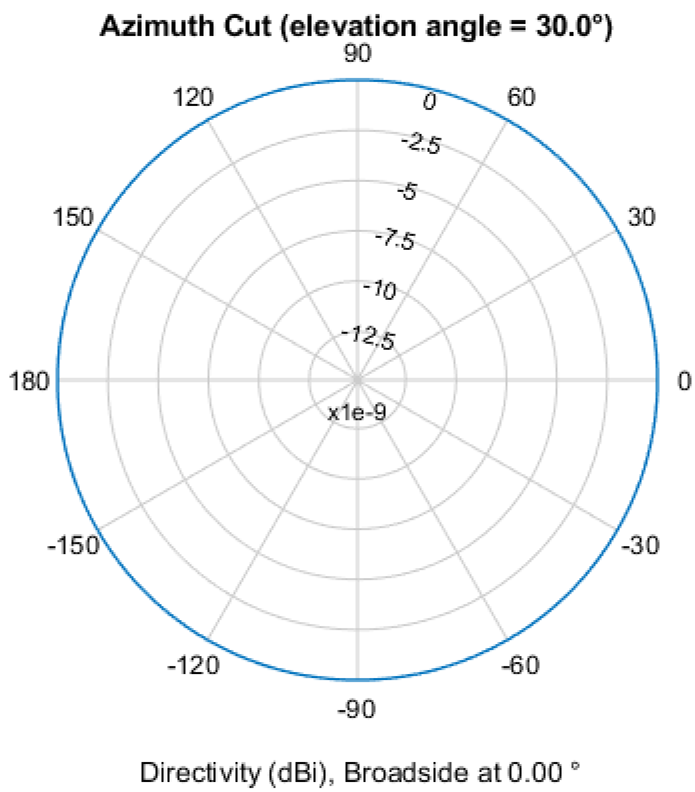


Azimuth Pattern of Isotropic Hydrophone

Examine the azimuth pattern of an isotropic hydrophone at 30° elevation. The frequency range is between 1 kHz and 10 kHz. Specify the voltage sensitivity as a vector.

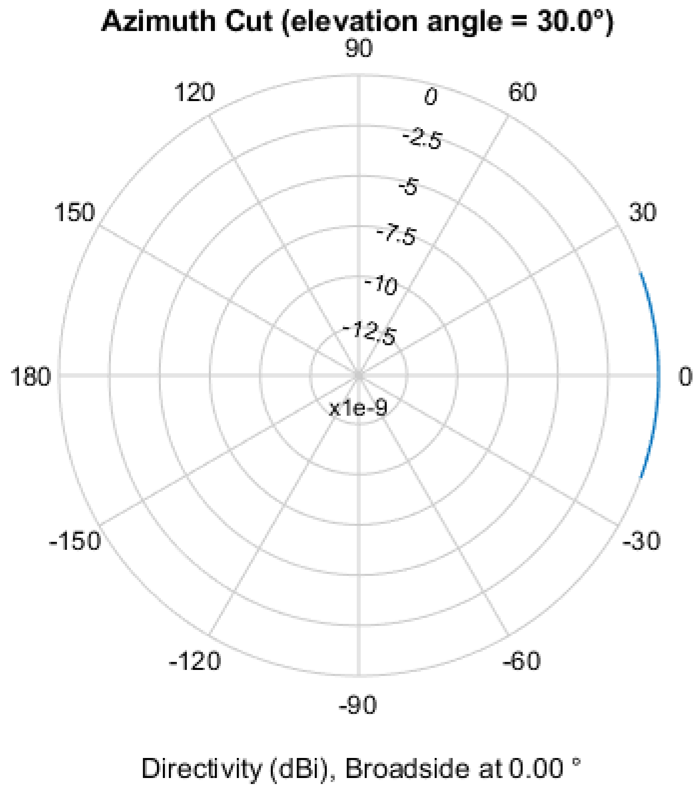
First, set up the hydrophone parameters.

```
fc = 3e3;
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1,10]*1e3, ...
    'VoltageSensitivity',[-100,-90,-100]);
patternAzimuth(hydrophone,fc,30)
```



Plot a smaller range of azimuth angles using the `Azimuth` parameter.

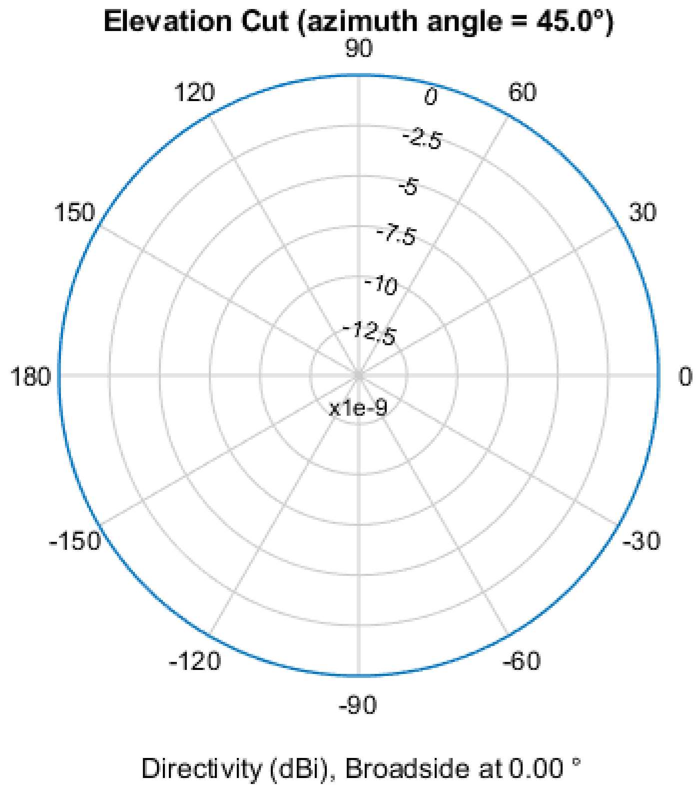
```
patternAzimuth(hydrophone,fc,30,'Azimuth',[-20:20])
```



Elevation Pattern of Isotropic Hydrophone

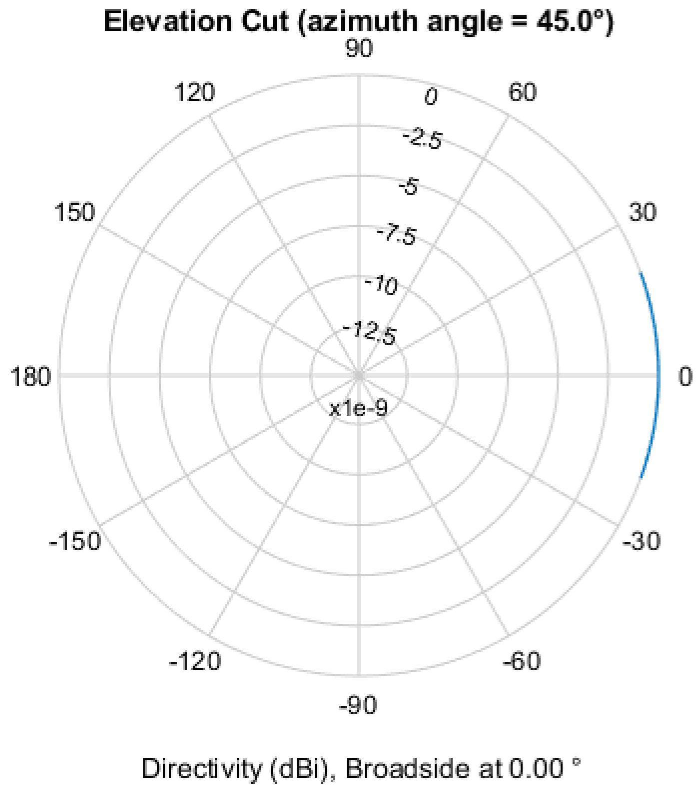
Plot an elevation cut of directivity of an isotropic hydrophone at 45° azimuth. Assume the signal frequency is 3 kHz. First, set up the hydrophone parameters.

```
fc = 3e3;
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1,10]*1e3, ...
    'VoltageSensitivity',[-100,-90,-100]);
patternElevation(hydrophone,fc,45)
```



Plot a smaller range of elevation angles using the `Elevation` parameter.

```
patternElevation(hydrophone,fc,45,'Elevation',-20:20)
```



More About

Hydrophone Sensitivity

Hydrophone sensitivity measures the response of a hydrophone to input sound pressure.

Hydrophone voltage sensitivity is the open circuit voltage (OCV) at the output of a hydrophone for a given input sound intensity. Another term for hydrophone sensitivity is open circuit receiving response (OCRR). Specifically, OCRR is the voltage generated by a hydrophone per µPa of sound intensity. OCRR is generally a function of frequency. If the sound intensity level (SIL) is expressed in dB//µPa and the output voltage is expressed in dB//1V, then OCRR is expressed in dB//1V/µPa. The output voltage of a hydrophone is related to the input sound level by

VdB = SIL + OCRR.

Consider a hydrophone that has OCRR = –160 dB//1V/µPa at 10 kHz. Assume that the SIL at the hydrophone due to a nearby ship is 120 dB//µPa. Then, the output voltage of the hydrophone is

VdB = SIL + OCRR = 120 dB + (–160) dB = –40 dB//1V.

In linear units,

$V = 10^{VdB/10} = 100\ \mu V$.

Algorithms

The total sensitivity of a hydrophone is a combination of its frequency sensitivity and spatial sensitivity. `phased.IsotropicHydrophone` calculates both sensitivities using nearest neighbor interpolation, and then multiplies the sensitivities to form the total sensitivity.

Version History

Introduced in R2017a

References

[1] Urick, R.J. *Principles of Underwater Sound.* 3rd Edition. New York: Peninsula Publishing, 1996.

[2] Sherman, C.S., and J. Butler. *Transducers and Arrays for Underwater Sound*. New York: Springer, 2007.

[3] Allen, J.B., and D. Berkely. “Image method for efficiently simulating small-room acoustics”, *Journal of the Acoustical Society of America*. Vol. 65, No. 4. April 1979, pp. 943–950.

[4] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002, pp. 274–304.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `pattern`, `patternAzimuth`, and `patternElevation` methods are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.ULA` | `phased.URA` | `phased.ConformalArray` | `phased.IsotropicProjector` | `phased.UnderwaterRadiatedNoise`

Topics
“Underwater Target Detection with an Active Sonar System”
“Locating an Acoustic Beacon with a Passive Sonar System”
Phased Array Gallery

directivity

System object: phased.IsotropicHydrophone
Package: phased

Directivity of isotropic hydrophone

Syntax

```
D = directivity(hydrophone,FREQ,ANGLE)
```

Description

D = directivity(hydrophone,FREQ,ANGLE) returns the "Directivity" on page 1-810 of the isotropic hydrophone, hydrophone, at frequencies specified by FREQ and in direction angles specified by ANGLE.

Input Arguments

hydrophone — Isotropic hydrophone
phased.IsotropicHydrophone System object

Isotropic hydrophone, specified as a phased.IsotropicHydrophone System object.

Example: phased.IsotropicHydrophone

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.
- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: [1e8 2e6]

Data Types: double

ANGLE — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If ANGLE is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, [az;el]. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Output Arguments

`D` — Directivity
M-by-*L* matrix

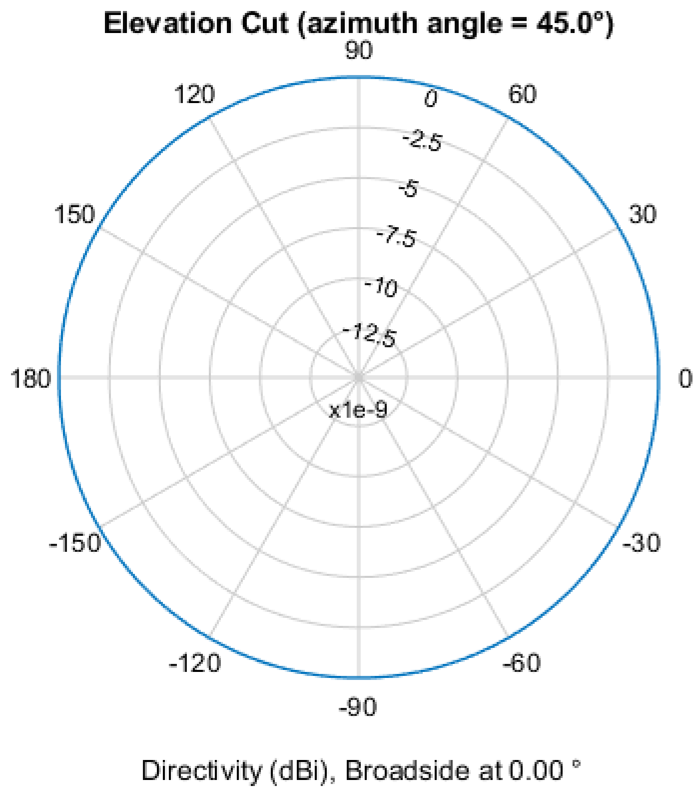
Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Isotropic Hydrophone

Compute the directivity of an isotropic hydrophone in different directions. Assume the signal frequency is 3 kHz. First, set up the hydrophone parameters.

```
fc = 3e3;
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1,10]*1e3, ...
    'VoltageSensitivity',[-100,-90,-100]);
patternElevation(hydrophone,fc,45)
```



First, select the angles of interest to be constant elevation angle at zero degrees. The five azimuth angles are centered around boresight (zero degrees azimuth and zero degrees elevation).

```
ang = [-20,-10,0,10,20; 0,0,0,0,0];
```

Compute the directivity along the constant elevation cut.

```
d = directivity(hydrophone,fc,ang)
```

```
d = 5×1

     0
     0
     0
     0
     0
```

The directivity of an isotropic hydrophone is zero in every direction.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on "Element Directivity" and "Array Directivity".

Version History

Introduced in R2017a

See Also

`pattern` | `patternAzimuth` | `patternElevation`

isPolarizationCapable

System object: `phased.IsotropicHydrophone`
Package: `phased`

Polarization capability

Syntax

`flag = isPolarizationCapable(hydrophone)`

Description

`flag = isPolarizationCapable(hydrophone)` returns a Boolean value, `flag`, indicating whether the `phased.IsotropicHydrophone` supports polarization. An element supports polarization if it can create or respond to polarized fields. This hydrophone does not support polarization.

Input Arguments

`hydrophone` — Isotropic hydrophone
`phased.IsotropicHydrophone` System object

Isotropic hydrophone, specified as a `phased.IsotropicHydrophone` System object.

Example: `phased.IsotropicHydrophone`

Output Arguments

`flag` — Polarization-capability flag
`true` | `false`

Polarization-capability returned as a Boolean value `true` if the hydrophone supports polarization or `false` if it does not. Because the `phased.IsotropicHydrophone` object does not support polarization, `flag` is always returned as `false`.

Version History

Introduced in R2017a

pattern

System object: `phased.IsotropicHydrophone`
Package: `phased`

Plot isotropic hydrophone directivity and patterns

Syntax

```
pattern(hydrophone,FREQ)
pattern(hydrophone,FREQ,AZ)
pattern(hydrophone,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(hydrophone,FREQ)` plots the 3D directivity pattern (in dBi) for the hydrophone, `hydrophone`. The operating frequency is specified in `FREQ`.

`pattern(hydrophone,FREQ,AZ)` plots the directivity pattern at the specified azimuth angle.

`pattern(hydrophone,FREQ,AZ,EL)` plots the directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the directivity pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the array pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Input Arguments

`hydrophone` — Isotropic hydrophone
`phased.IsotropicHydrophone` System object

Isotropic hydrophone, specified as a `phased.IsotropicHydrophone` System object.

Example: `phased.IsotropicHydrophone`

`FREQ` — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element.

Otherwise, the element produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`AZ` — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

`EL` — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Normalize` — Display normalize pattern
`true` (default) | `false`

Display normalized pattern, specified as the comma-separated pair consisting of `'Normalize`' and a Boolean. Set this parameter to `true` to display a normalized pattern. This parameter does not apply when you set `'Type'` to `'directivity'`. Directivity patterns are already normalized.

Data Types: `logical`

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

Output Arguments

PAT — Element pattern
N-by-*M* real-valued matrix

Element pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of PAT correspond to the azimuth angles in the vector specified by EL_ANG. The columns correspond to the elevation angles in the vector specified by AZ_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

Examples

Response and Pattern of Isotropic Hydrophone for Single Frequency

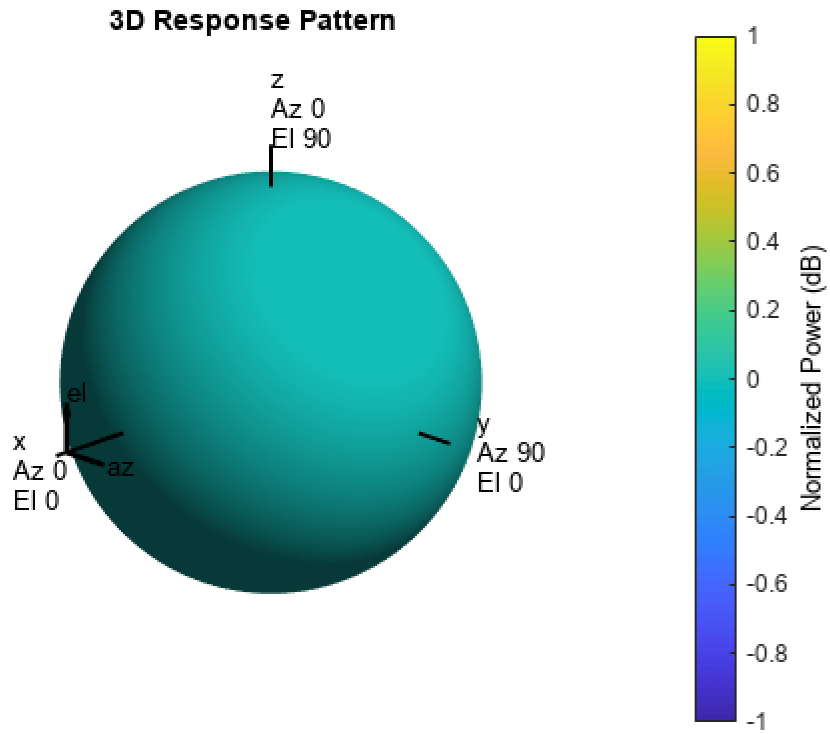
Examine the response and patterns of an isotropic hydrophone operating between 1 kHz and 10 kHz.

Set up the hydrophone parameters. Obtain the voltage sensitivity at five different elevation angles: -30°, -15°, 0°, 15° and 30°. All azimuth angles are at 0°. The sensitivities are computed at the signal frequency of 2 kHz.

```
hydrophone = phased.IsotropicHydrophone('FrequencyRange', ...
    [1 10]*1e3);
fc = 2e3;
resp = hydrophone(fc,[0 0 0 0 0;-30 -15 0 15 30]);
```

Draw a 3-D plot of the voltage sensitivity.

```
pattern(hydrophone,fc,[-180:180],[-90:90], ...
    'CoordinateSystem','polar','Type','powerdb')
```



Response and Pattern of Isotropic Hydrophone at Multiple Frequencies

Examine the response and patterns of an isotropic hydrophone at three different frequencies. The hydrophone operates between 1 kHz and 10 kHz. Specify the voltage sensitivity as a vector.

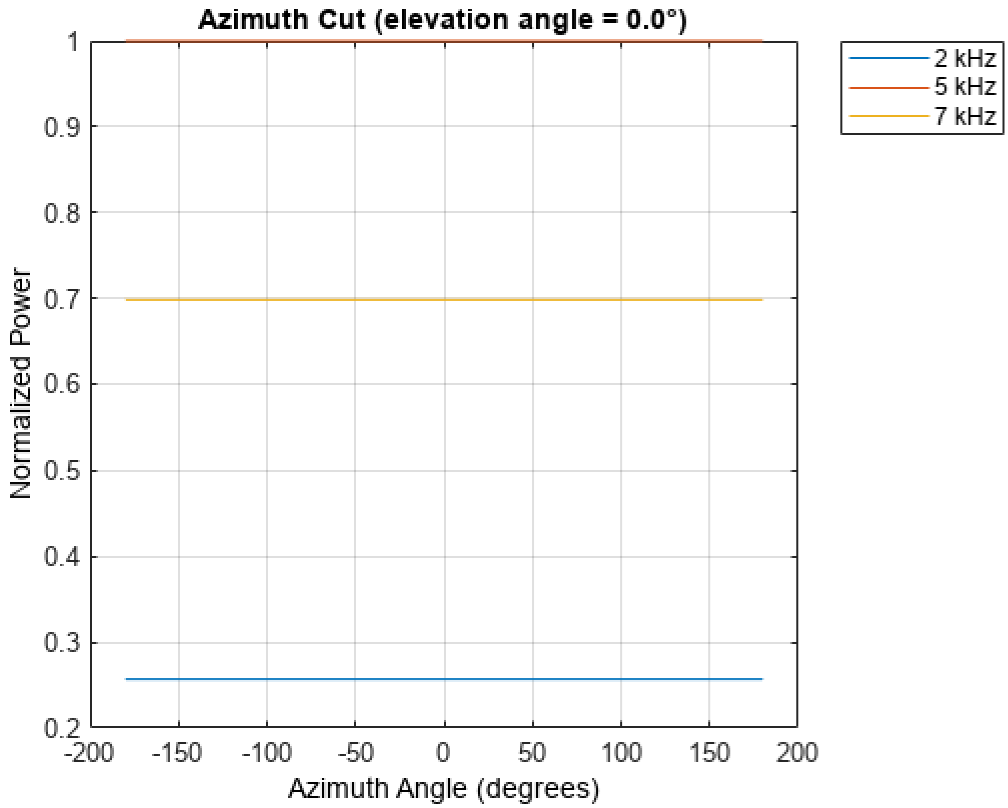
Set up the hydrophone parameters and obtain the voltage sensitivity at 45° azimuth and 30° elevation. Compute the sensitivities at the signal frequencies of 2, 5, and 7 kHz.

```
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1 10]*1e3, ...
    'VoltageSensitivity',[-100 -90 -100]);
fc = [2e3 5e3 7e3];
resp = hydrophone(fc,[45;30])
```

```
resp = 1×3

   14.8051   29.2202   24.4152
```

Draw a 2-D plot of the voltage sensitivity as a function of azimuth.

```
pattern(hydrophone,fc,[-180:180],0,'CoordinateSystem','rectangular',...
    'Type','power')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2017a

See Also

`patternAzimuth` | `patternElevation`

patternAzimuth

System object: `phased.IsotropicHydrophone`
Package: `phased`

Plot isotropic hydrophone directivity and response patterns versus azimuth

Syntax

```
patternAzimuth(hydrophone,FREQ)
patternAzimuth(hydrophone,FREQ,EL)
patternAzimuth(hydrophone,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(hydrophone,FREQ)` plots the 2-D element directivity pattern versus azimuth (in dBi) for the element `hydrophone` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

`patternAzimuth(hydrophone,FREQ,EL)`, in addition, plots the 2-D element directivity pattern versus azimuth (in dBi) at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(hydrophone,FREQ,EL,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

hydrophone — Isotropic hydrophone

`phased.IsotropicHydrophone` System object

Isotropic hydrophone, specified as a `phased.IsotropicHydrophone` System object.

Example: `phased.IsotropicHydrophone`

FREQ — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

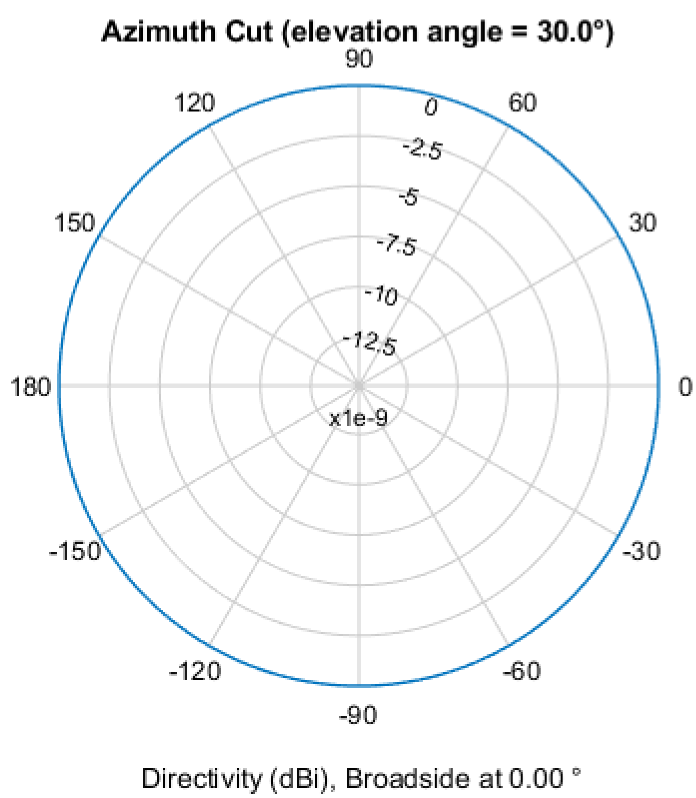
Examples

Azimuth Pattern of Isotropic Hydrophone

Examine the azimuth pattern of an isotropic hydrophone at 30° elevation. The frequency range is between 1 kHz and 10 kHz. Specify the voltage sensitivity as a vector.

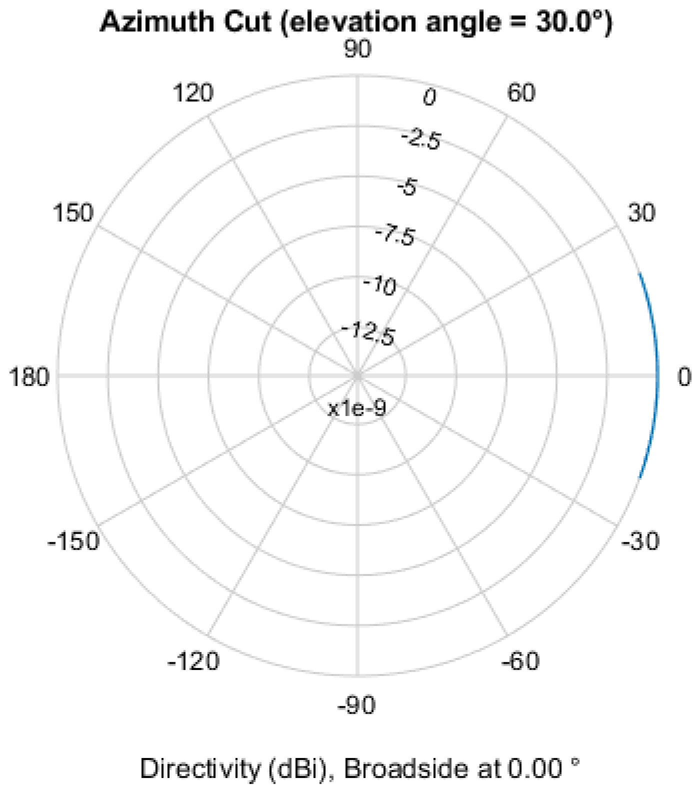
First, set up the hydrophone parameters.

```
fc = 3e3;
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1,10]*1e3, ...
    'VoltageSensitivity',[-100,-90,-100]);
patternAzimuth(hydrophone,fc,30)
```



Plot a smaller range of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(hydrophone,fc,30,'Azimuth',[-20:20])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2017a

See Also

`pattern` | `patternElevation`

patternElevation

System object: `phased.IsotropicHydrophone`
Package: `phased`

Plot isotropic hydrophone directivity and response patterns versus elevation

Syntax

```
patternElevation(hydrophone,FREQ)
patternElevation(hydrophone,FREQ,AZ)
patternElevation(hydrophone,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(hydrophone,FREQ)` plots the 2-D element directivity pattern versus elevation (in dBi) for the element `hydrophone` at zero degrees azimuth angle. The argument `FREQ` specifies the operating frequency.

`patternElevation(hydrophone,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(hydrophone,FREQ,AZ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

hydrophone — Isotropic hydrophone
`phased.IsotropicHydrophone` System object

Isotropic hydrophone, specified as a `phased.IsotropicHydrophone` System object.

Example: `phased.IsotropicHydrophone`

FREQ — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: 1e8

Data Types: double

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: [0,10,20]

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Elevation — Elevation angles
[-90:90] (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of 'Elevation' and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: 'Elevation',[-90:2:90]

Data Types: double

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

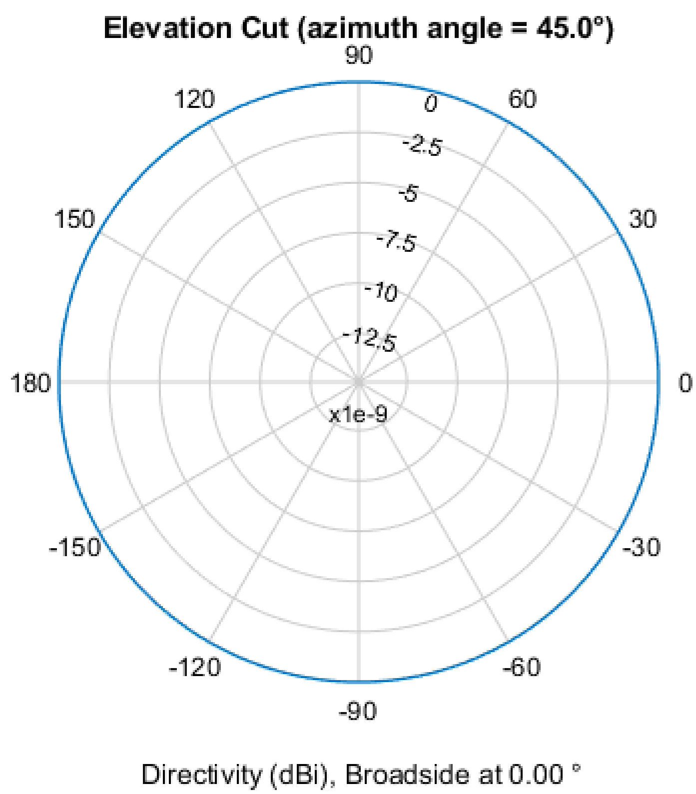
Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Elevation Pattern of Isotropic Hydrophone

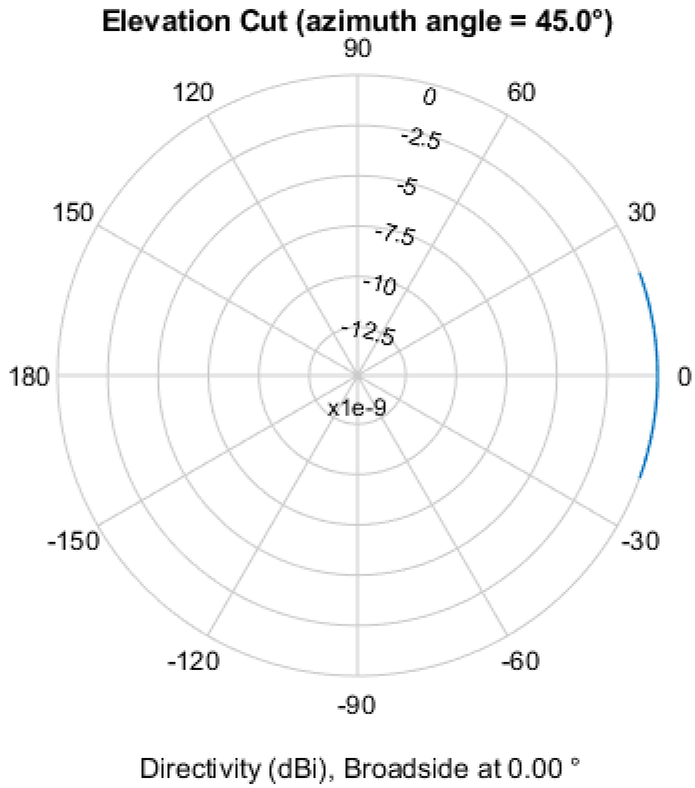
Plot an elevation cut of directivity of an isotropic hydrophone at 45° azimuth. Assume the signal frequency is 3 kHz. First, set up the hydrophone parameters.

```
fc = 3e3;
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1,10]*1e3, ...
    'VoltageSensitivity',[-100,-90,-100]);
patternElevation(hydrophone,fc,45)
```



Plot a smaller range of elevation angles using the `Elevation` parameter.

```
patternElevation(hydrophone,fc,45,'Elevation',-20:20)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2017a

See Also
`pattern` | `patternAzimuth`

step

System object: `phased.IsotropicHydrophone`
Package: `phased`

Voltage sensitivity of isotropic hydrophone

Syntax

`resp = step(hydrophone,freq,ang)`

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`resp = step(hydrophone,freq,ang)` returns the voltage sensitivity for the hydrophone at the specified operating frequencies and in the specified directions of arriving signals.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`hydrophone` — Isotropic hydrophone
`phased.IsotropicHydrophone` System object

Isotropic hydrophone, specified as a `phased.IsotropicHydrophone` System object.

Example: `phased.IsotropicHydrophone`

`freq` — Voltage sensitivity frequencies
positive real scalar | real-valued 1-by-L vector of positive values

Voltage sensitivity frequencies of hydrophone, specified as a positive real scalar or a real-valued 1-by-L vector of positive values. Units are in Hz.

Data Types: `double`

`ang` — Direction of arriving signals
real-valued 1-by-M row vector | real-valued 2-by-M matrix

Direction of arriving signals, specified as a real-valued 1-by-M row vector or 2-by-M matrix. When `ang` is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth;elevation]. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

When ang is a 1-by-*M* row vector, each element specifies the azimuth angle of the arriving signal. In this case, the corresponding elevation angle is assumed to be zero.

Data Types: double

Output Arguments

resp — Voltage sensitivity of hydrophone

Voltage sensitivity of hydrophone, returned as a real-valued *M*-by-*L* matrix. *M* represents the number of angles specified in ang, and *L* represents the number of frequencies specified in freq. Units are in V/Pa.

Examples

Response and Pattern of Isotropic Hydrophone for Single Frequency

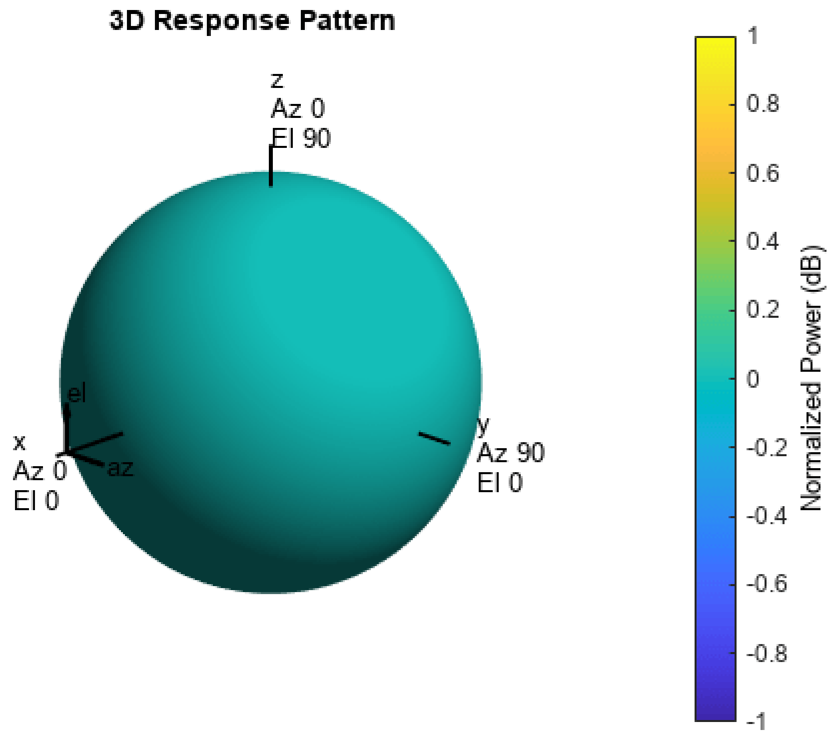
Examine the response and patterns of an isotropic hydrophone operating between 1 kHz and 10 kHz.

Set up the hydrophone parameters. Obtain the voltage sensitivity at five different elevation angles: -30°, -15°, 0°, 15° and 30°. All azimuth angles are at 0°. The sensitivities are computed at the signal frequency of 2 kHz.

```
hydrophone = phased.IsotropicHydrophone('FrequencyRange', ...
    [1 10]*1e3);
fc = 2e3;
resp = hydrophone(fc,[0 0 0 0 0;-30 -15 0 15 30]);
```

Draw a 3-D plot of the voltage sensitivity.

```
pattern(hydrophone,fc,[-180:180],[-90:90], ...
    'CoordinateSystem','polar','Type','powerdb')
```



Response and Pattern of Isotropic Hydrophone at Multiple Frequencies

Examine the response and patterns of an isotropic hydrophone at three different frequencies. The hydrophone operates between 1 kHz and 10 kHz. Specify the voltage sensitivity as a vector.

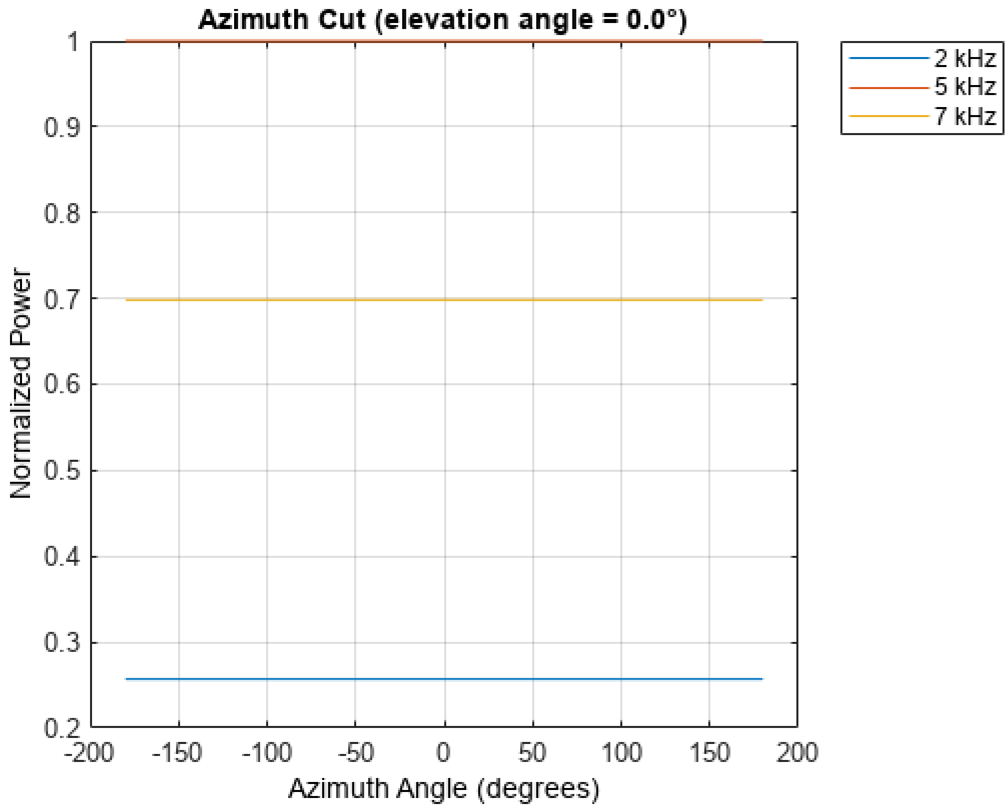
Set up the hydrophone parameters and obtain the voltage sensitivity at 45° azimuth and 30° elevation. Compute the sensitivities at the signal frequencies of 2, 5, and 7 kHz.

```
hydrophone = phased.IsotropicHydrophone('FrequencyRange',[1 10]*1e3, ...
    'VoltageSensitivity',[-100 -90 -100]);
fc = [2e3 5e3 7e3];
resp = hydrophone(fc,[45;30])
```

```
resp = 1×3

   14.8051   29.2202   24.4152
```

Draw a 2-D plot of the voltage sensitivity as a function of azimuth.

```
pattern(hydrophone,fc,[-180:180],0,'CoordinateSystem','rectangular',...
    'Type','power')
```



Algorithms

The total sensitivity of a hydrophone is a combination of its frequency sensitivity and spatial sensitivity. `phased.IsotropicHydrophone` calculates both sensitivities using nearest neighbor interpolation, and then multiplies the sensitivities to form the total sensitivity.

See Also

`uv2azel` | `phitheta2azel`

phased.IsotropicProjector

Package: `phased`

Isotropic projector element

Description

The `phased.IsotropicProjector` System object creates an isotropic sound projector used in sonar applications. An isotropic projector has the same response in all directions. The response is the radiated sound intensity per unit input voltage to the projector. You can change the response using the `VoltageResponse` property.

To compute the response of the isotropic projector for specified directions:

1. Create the `phased.IsotropicProjector` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
projector = phased.IsotropicProjector
projector = phased.IsotropProjector(Name=Value)
```

Description

`projector = phased.IsotropicProjector` creates an isotropic projector System object, `projector`.

`projector = phased.IsotropProjector(Name=Value)` creates an Isotropic projector object, `projector`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1=Value1`,...,`NameN=ValueN`).

Example: `projector = phased.IsotropicProjector(FrequencyRange=[0 1000],BackBaffled=true)` creates an isotropic projector element with its frequency range specified between 0 and 1000 Hz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`FrequencyRange` — Operating frequency range of projector
`[0 100e6]` (default) | real-valued 1-by-2 vector

Operating frequency range of projector, specified as a 1-by-2 row vector in the form of `[LowerBound HigherBound]`. This property defines the frequency range over which the projector has a response. The projector element has zero response outside this frequency range. Units are in Hz..

Example: `[0 10e3]`

Data Types: `double`

`VoltageResponse` — Voltage response of projector
`120` (default) | scalar | real-valued 1-by-*K* row vector

Voltage response of projector, specified as a scalar or real-valued 1-by-*K* row vector. When you specify voltage response as a scalar, that value applies to the entire frequency range specified by `FrequencyRange`. When you specify the voltage sensitivity as a vector, the frequency range is divided into K-1 equal intervals. The response values are assigned to the interval end points. Then, the `step` method interpolates the voltage response for any frequency inside the frequency range. Units are in dB ref: 1 µPa/V. See “Projector Voltage Response” on page 1-844 for more details.

Example: `10`

Data Types: `double`

`BackBaffled` — Back baffle response of projector
`false` (default) | `true`

Back baffle response of projector, specified as `false` or `true`. Set this property to `true` to back baffle the projector response. When the projector is back baffled, the projector response for all azimuth angles beyond ±90 from broadside are zero. Broadside is defined as 0° azimuth and 0° elevation.

When the value of this property is `false`, the projector is not back baffled.

Usage

Syntax

```
resp = projector(freq,ang)
```

Description

`resp = projector(freq,ang)` returns the voltage response `resp` for the projector at the specified operating frequencies `freq` and in the specified directions `ang` of arriving signals.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

freq — Voltage response frequencies
positive real scalar | real-valued 1-by-*L* vector of positive values

Voltage response frequencies of projector, specified as a positive real scalar or a real-valued 1-by-*L* vector of positive values. Units are in Hz.

Data Types: `double`

ang — Direction of arriving signals
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Direction of arriving signals, specified as a real-valued 1-by-*M* row vector or 2-by-*M* matrix. When `ang` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

When `ang` is a 1-by-*M* row vector, each element specifies the azimuth angle of the arriving signal. In this case, the corresponding elevation angle is assumed to be zero.

Data Types: `double`

Output Arguments

resp — Voltage response of projector
real-valued *M*-by-*L* matrix

Voltage response of projector, returned as a real-valued *M*-by-*L* matrix. *M* represents the number of angles specified in `ang`, and *L* represents the number of frequencies specified in `freq`. Units are in V/Pa.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Response of Isotropic Projector

Find the response of an isotropic projector with default properties. Obtain the voltage response at five different elevation angles: -30°, -15°, 0°, 15° and 30°. All azimuth angles are set to 0°. The voltage response is computed at 2 kHz.

```
projector = phased.IsotropicProjector;
fc = 2e3;
resp = projector(fc,[0,0,0,0,0;-30,-15,0,15,30])
```

```
resp = 5×1

     1
     1
     1
     1
     1
```

Response and Pattern of Isotropic Projector at Single Frequency

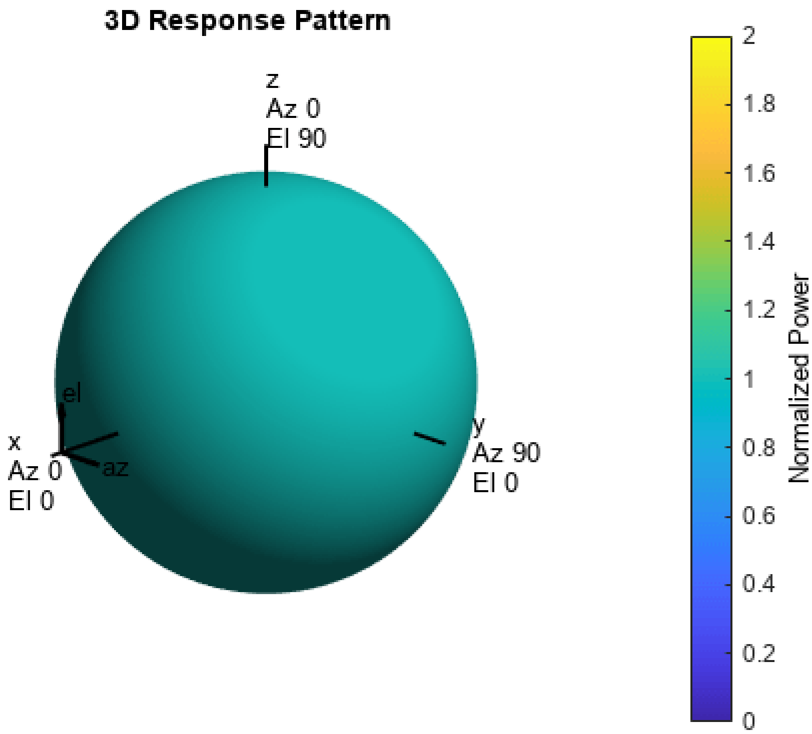
Examine the response and patterns of an isotropic projector operating between 1 kHz and 10 kHz.

Set the projector parameters and obtain the voltage response at five different elevation angles: -30°, -15°, 0°, 15° and 30°. All elevation angles at 0° azimuth angle. The voltage response is computed at 2 kHz.

```
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3);
fc = 2e3;
resp = projector(fc,[0,0,0,0,0;-30,-15,0,15,30]);
```

Draw a 3-D plot of the voltage response.

```
pattern(projector,fc,[-180:180],[-90:90],'CoordinateSystem','polar', ...
    'Type','power')
```



Plot Beamwidth of Isotropic Projector

Find the beamwidth of an isotropic underwater projector at 10 kHz.

First, create an isotropic projector element.

```
projector = phased.IsotropicProjector('FrequencyRange',[10 12000])
```

```
projector = 
  phased.IsotropicProjector with properties:

    VoltageResponse: 120
     FrequencyRange: [10 12000]
        BackBaffled: false
```

Then, use the beamwidth object function to plot the 3 dB beamwidth of the projector.

```
[bmw,angs] = beamwidth(projector,10000,'dbDown',3)
```

```
bmw = 360
```

```
angs = 1×2

  -180   180
```

Because the projector is isotropic, there is no 3 dB down point.

Response and Pattern of Isotropic Projector at Multiple Frequencies

Examine the response and patterns of an isotropic projector at three different frequencies. The projector operates between 1 kHz and 10 kHz. Specify the voltage response as a vector.

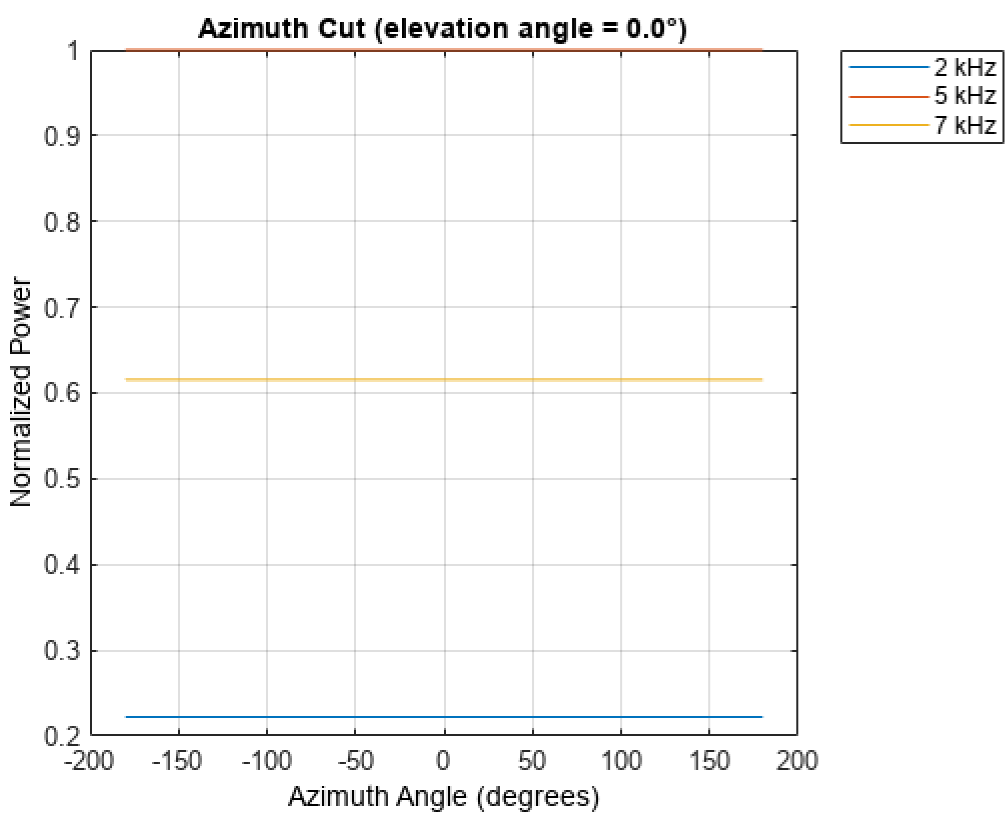
Set up the projector parameters, and obtain the voltage response at 45° azimuth and 30° elevation. Compute the responses at signal frequencies of 2, 5, and 7 kHz.

```
projector = phased.IsotropicProjector('FrequencyRange',[1 10]*1e3, ...
    'VoltageResponse',[90 95 100 95 90]);
fc = [2e3 5e3 7e3];
resp = projector(fc,[45;30]);
resp
```

```
resp = 1×3

    0.0426    0.0903    0.0708
```

Next, draw a 2-D plot of the voltage response as a function of azimuth.

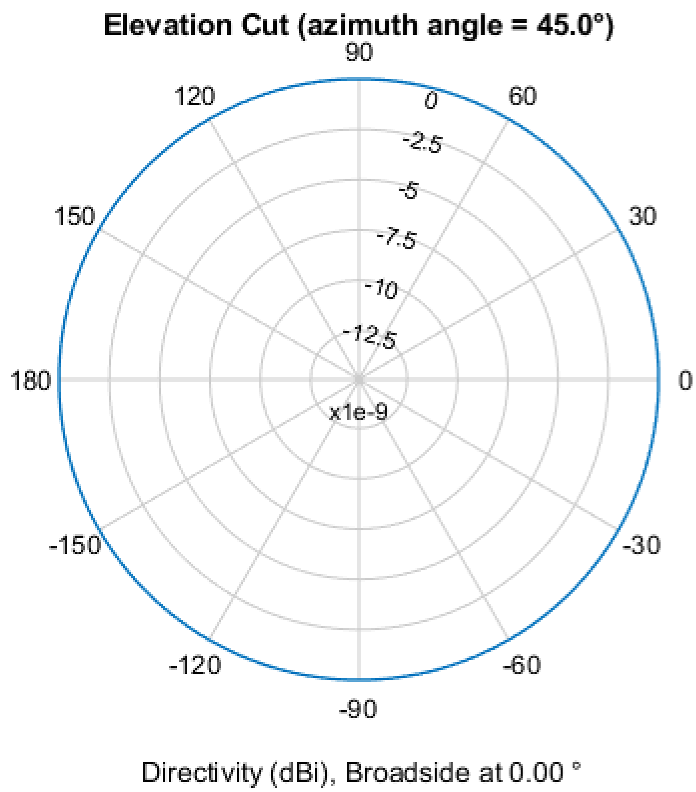
```
pattern(projector,fc,[-180:180],0,'CoordinateSystem','rectangular', ...
    'Type','power')
```



Directivity of Isotropic Projector

Compute the directivity of an isotropic projector in different directions. Assume the signal frequency is 3 kHz. First, set the projector parameters.

```
fc = 3e3;
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3, ...
    'VoltageResponse',[100,110,120,110,100]);
patternElevation(projector,fc,45)
```



Select the angles of interest to be constant elevation angle at zero degrees. The five azimuth angles are centered around boresight (zero degrees azimuth and zero degrees elevation).

```
ang = [-20,-10,0,10,20; 0,0,0,0,0];
```

Compute the directivity along the constant elevation cut.

```
d = directivity(projector,fc,ang)
```

```
d = 5×1

     0
     0
     0
     0
```

```
0
```

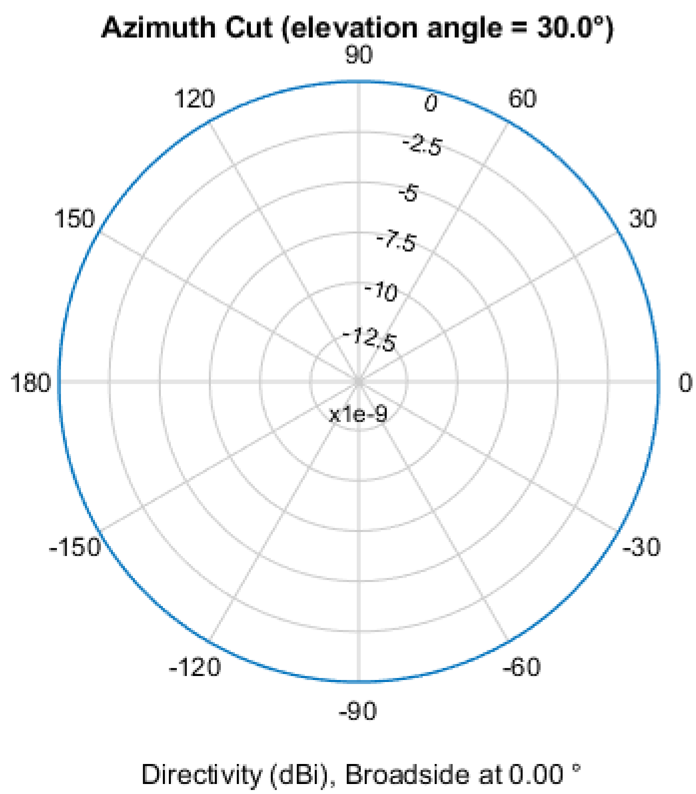
The directivity of an isotropic projector is zero in every direction.

Azimuth Pattern of Isotropic Projector

Examine the azimuth pattern of an isotropic projector at 30° elevation. The frequency range is between 1 kHz and 10 kHz. Specify the voltage response as a scalar.

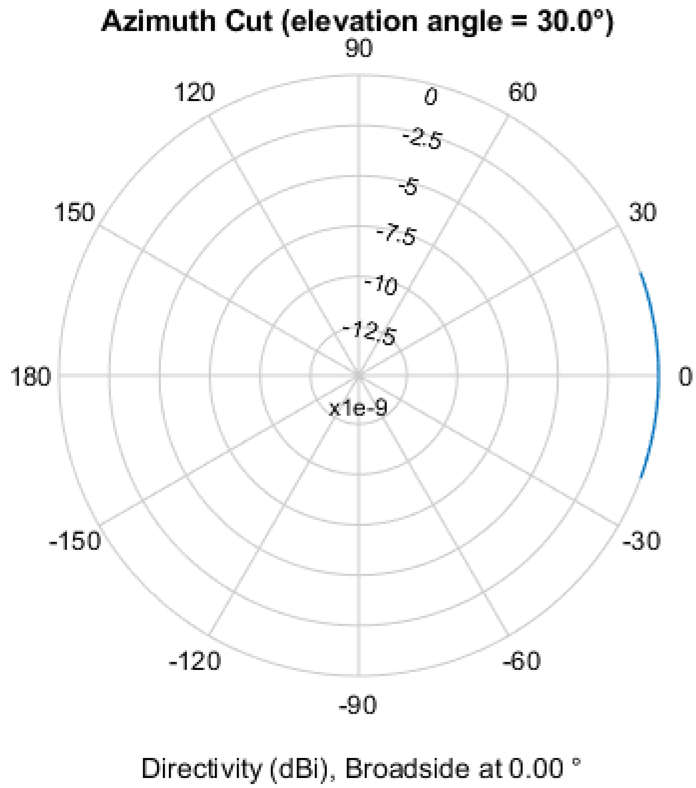
Set the projector parameters.

```
fc = 3e3;
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3, ...
    'VoltageResponse',-115);
patternAzimuth(projector,fc,30)
```



Plot a smaller range of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(projector,fc,30,'Azimuth',[-20:20])
```

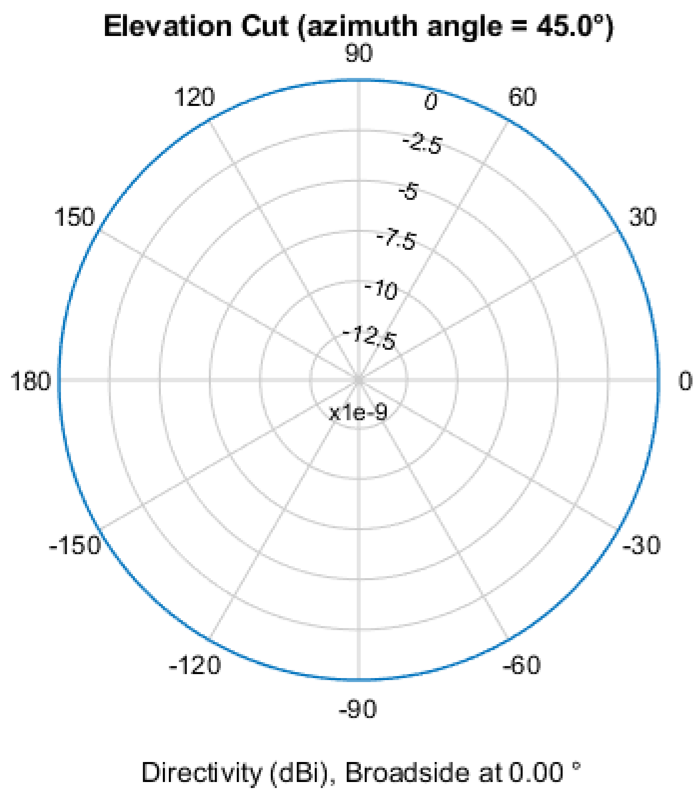


Elevation Pattern of Isotropic Projector

Plot an elevation cut of the directivity of an isotropic projector at 45° azimuth. Assume the signal frequency is 3 kHz.

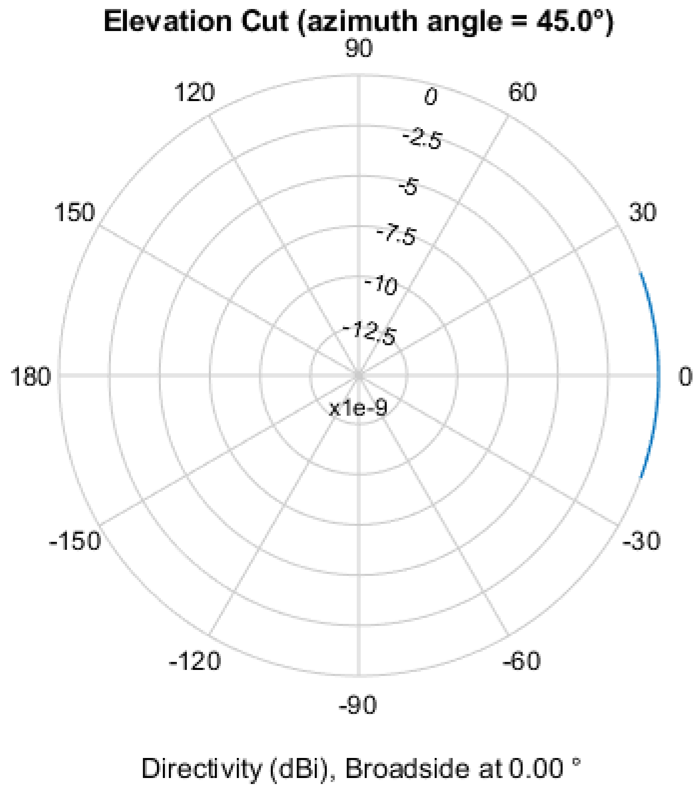
Create the isotropic projector object and call the pattern object function.

```
fc = 3e3;
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3, ...
    'VoltageResponse',70);
patternElevation(projector,fc,45)
```



Plot a smaller range of elevation angles using the `Elevation` parameter.

```
patternElevation(projector,fc,45,'Elevation',-20:20)
```



More About

Projector Voltage Response

The voltage response of a projector relates the transmitted sound intensity to the input voltage.

For a sound projector, the transmitting voltage response (TVR) is the sound intensity in µPa per volt, when measured at one meter from the projector. TVR is generally a function of frequency. If the sound intensity level (SIL) is expressed in dB//µPa and the output voltage (VdB) is expressed in dB//1V, then TVR is expressed in dB//µPa/1V. The output sound pressure level of a hydrophone is related to the input voltage level by

SIL = TVR + VdB.

Consider a projector that has TVR = 160 dB//µPa/1V at 10 kHz. If the projector input voltage is 200 V, then the VdB is 23 dB and the sound intensity level (SIL) at one meter is

SIL = TVR + VdB = 160 + 23 = 173 dB//µPa.

Version History

Introduced in R2017a

References

[1] Urick, R.J. *Principles of Underwater Sound.* 3rd Edition. New York: Peninsula Publishing, 1996.

[2] Sherman, C.S., and J. Butler. *Transducers and Arrays for Underwater Sound*. New York: Springer, 2007.

[3] Allen, J.B., and D. Berkely. “Image method for efficiently simulating small-room acoustics”, *Journal of the Acoustical Society of America*. Vol. 65, No. 4. April 1979, , pp. 943–950.

[4] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002, pp. 274–304.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `pattern`, `patternAzimuth`, and `patternElevation` methods are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also
`phased.ULA` | `phased.URA` | `phased.ConformalArray` | `phased.IsotropicHydrophone` | `phased.UnderwaterRadiatedNoise`

Topics
“Underwater Target Detection with an Active Sonar System”
“Locating an Acoustic Beacon with a Passive Sonar System”
Phased Array Gallery

directivity

System object: phased.IsotropicProjector
Package: phased

Directivity of isotropic projector

Syntax

```
D = directivity(projector,FREQ,ANGLE)
```

Description

D = directivity(projector,FREQ,ANGLE) returns the "Directivity" on page 1-848 of the isotropic projector, projector, at frequencies specified by FREQ and in the directions specified by ANGLE.

Input Arguments

projector — Isotropic projector

phased.IsotropicProjector System object

Isotropic projector, specified as a phased.IsotropicProjector System object.

Example: phased.IsotropicProjector

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.
- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: [1e8 2e6]

Data Types: double

ANGLE — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If ANGLE is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, [az;el]. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: `[45 60; 0 10]`

Data Types: `double`

Output Arguments

`D` — Directivity
M-by-*L* matrix

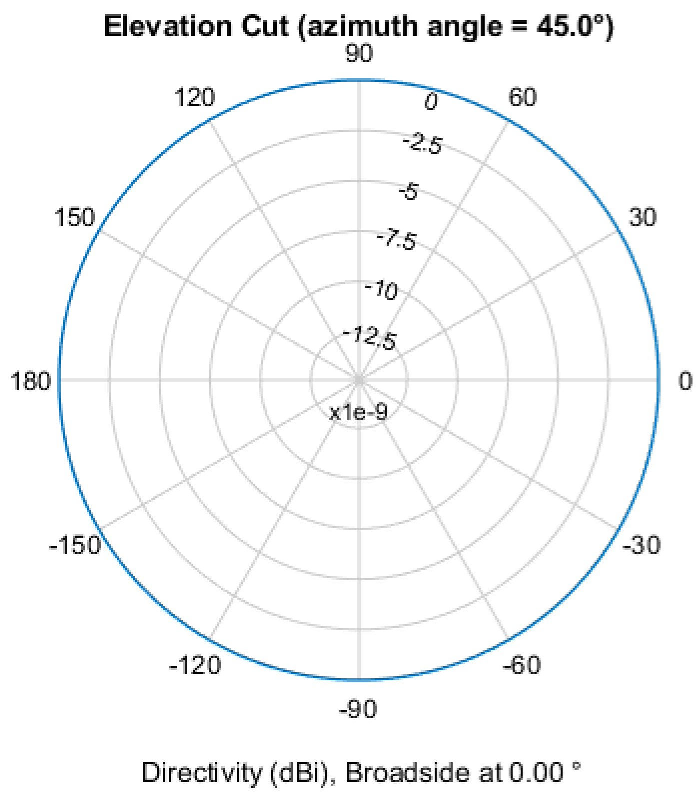
Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Isotropic Projector

Compute the directivity of an isotropic projector in different directions. Assume the signal frequency is 3 kHz. First, set the projector parameters.

```
fc = 3e3;
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3, ...
    'VoltageResponse',[100,110,120,110,100]);
patternElevation(projector,fc,45)
```



Select the angles of interest to be constant elevation angle at zero degrees. The five azimuth angles are centered around boresight (zero degrees azimuth and zero degrees elevation).

```
ang = [-20,-10,0,10,20; 0,0,0,0,0];
```

Compute the directivity along the constant elevation cut.

```
d = directivity(projector,fc,ang)
```

```
d = 5×1

     0
     0
     0
     0
     0
```

The directivity of an isotropic projector is zero in every direction.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2017a

See Also

`pattern` | `patternAzimuth` | `patternElevation`

isPolarizationCapable

System object: `phased.IsotropicProjector`
Package: `phased`

Polarization capability

Syntax

```
flag = isPolarizationCapable(projector)
```

Description

`flag = isPolarizationCapable(projector)` returns a Boolean value, `flag`, indicating whether the `phased.IsotropicProjector` supports polarization. An element supports polarization if it can create or respond to polarized fields. This projector does not support polarization.

Input Arguments

`projector` — Isotropic projector
`phased.IsotropicProjector` System object

Isotropic projector, specified as a `phased.IsotropicProjector` System object.

Example: `phased.IsotropicProjector`

Output Arguments

`flag` — Polarization-capability flag
`true` | `false`

Polarization-capability returned as a Boolean value `true` if the projector supports polarization or `false` if it does not. Because the `phased.IsotropicProjector` object does not support polarization, `flag` is always returned as `false`.

Version History

Introduced in R2017a

pattern

System object: phased.IsotropicProjector
Package: phased

Plot isotropic projector directivity and patterns

Syntax

```
pattern(projector,FREQ)
pattern(projector,FREQ,AZ)
pattern(projector,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(projector,FREQ)` plots the 3D directivity pattern (in dBi) for the projector specified in `projector`. The operating frequency is specified in `FREQ`.

`pattern(projector,FREQ,AZ)` plots the projector directivity pattern at the specified azimuth angle.

`pattern(projector,FREQ,AZ,EL)` plots the projector directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the projector pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the projector pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Input Arguments

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`AZ` — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

`EL` — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

`Type` — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

PlotStyle — Plotting style
'overlay' (default) | 'waterfall'

Plotting style, specified as the comma-separated pair consisting of 'Plotstyle' and either 'overlay' or 'waterfall'. This parameter applies when you specify multiple frequencies in FREQ in 2-D plots. You can draw 2-D plots by setting one of the arguments AZ or EL to a scalar.

Data Types: char

Polarization — Polarized field component
'combined' (default) | 'H' | 'V'

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and 'combined', 'H', or 'V'. This parameter applies only when the sensors are polarization-capable and when the 'Type' parameter is not set to 'directivity'. This table shows the meaning of the display options.

'Polarization'	Display
'combined'	Combined *H* and *V* polarization components
'H'	*H* polarization component
'V'	*V* polarization component

Example: 'V'

Data Types: char

Output Arguments

PAT — Element pattern
N-by-*M* real-valued matrix

Element pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of PAT correspond to the azimuth angles in the vector specified by EL_ANG. The columns correspond to the elevation angles in the vector specified by AZ_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

Examples

Response and Pattern of Isotropic Projector at Single Frequency

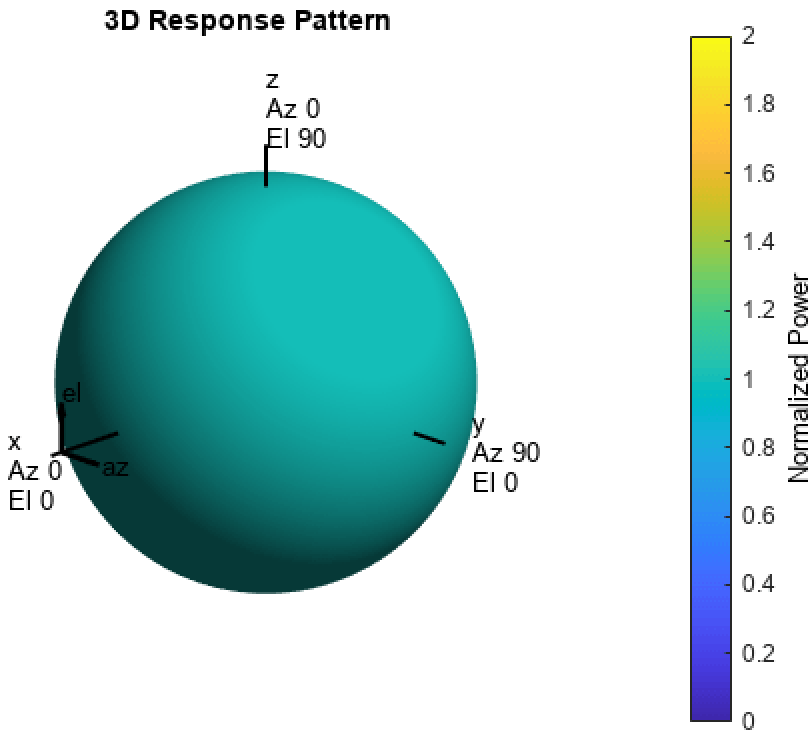
Examine the response and patterns of an isotropic projector operating between 1 kHz and 10 kHz.

Set the projector parameters and obtain the voltage response at five different elevation angles: -30°, -15°, 0°, 15° and 30°. All elevation angles at 0° azimuth angle. The voltage response is computed at 2 kHz.

```
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3);
fc = 2e3;
resp = projector(fc,[0,0,0,0,0;-30,-15,0,15,30]);
```

Draw a 3-D plot of the voltage response.

```
pattern(projector,fc,[-180:180],[-90:90],'CoordinateSystem','polar', ...
    'Type','power')
```



Response and Pattern of Isotropic Projector at Multiple Frequencies

Examine the response and patterns of an isotropic projector at three different frequencies. The projector operates between 1 kHz and 10 kHz. Specify the voltage response as a vector.

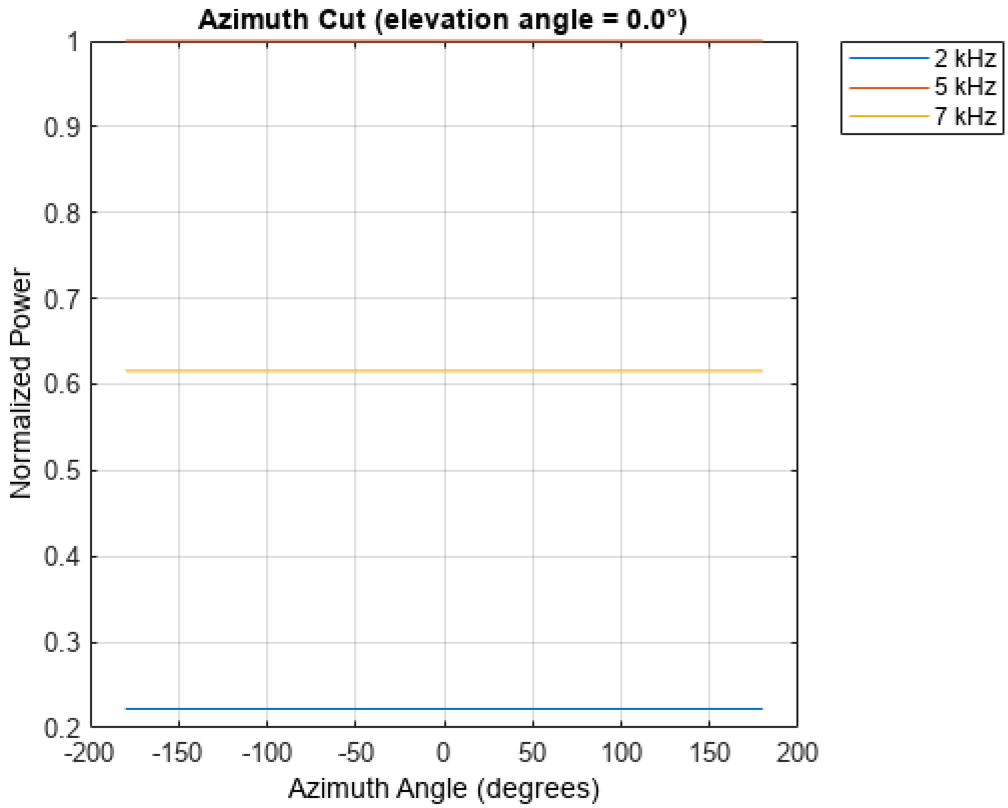
Set up the projector parameters, and obtain the voltage response at 45° azimuth and 30° elevation. Compute the responses at signal frequencies of 2, 5, and 7 kHz.

```
projector = phased.IsotropicProjector('FrequencyRange',[1 10]*1e3, ...
    'VoltageResponse',[90 95 100 95 90]);
fc = [2e3 5e3 7e3];
resp = projector(fc,[45;30]);
resp
```

```
resp = 1×3

    0.0426    0.0903    0.0708
```

Next, draw a 2-D plot of the voltage response as a function of azimuth.

```
pattern(projector,fc,[-180:180],0,'CoordinateSystem','rectangular', ...
    'Type','power')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2017a

See Also
`patternAzimuth` | `patternElevation`

patternAzimuth

System object: phased.IsotropicProjector
Package: phased

Plot isotropic projector directivity and response patterns versus azimuth

Syntax

```
patternAzimuth(projector,FREQ)
patternAzimuth(projector,FREQ,EL)
patternAzimuth(projector,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(projector,FREQ)` plots the 2-D element directivity pattern versus azimuth (in dBi) for the projector, `projector,` at zero-degrees elevation angle. The argument `FREQ` specifies the operating frequency.

`patternAzimuth(projector,FREQ,EL)`, in addition, plots the 2-D element directivity pattern versus azimuth (in dBi) at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(projector,FREQ,EL,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`projector` — Isotropic projector
`phased.IsotropicProjector` System object

Isotropic projector, specified as a `phased.IsotropicProjector` System object.

Example: `phased.IsotropicProjector`

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

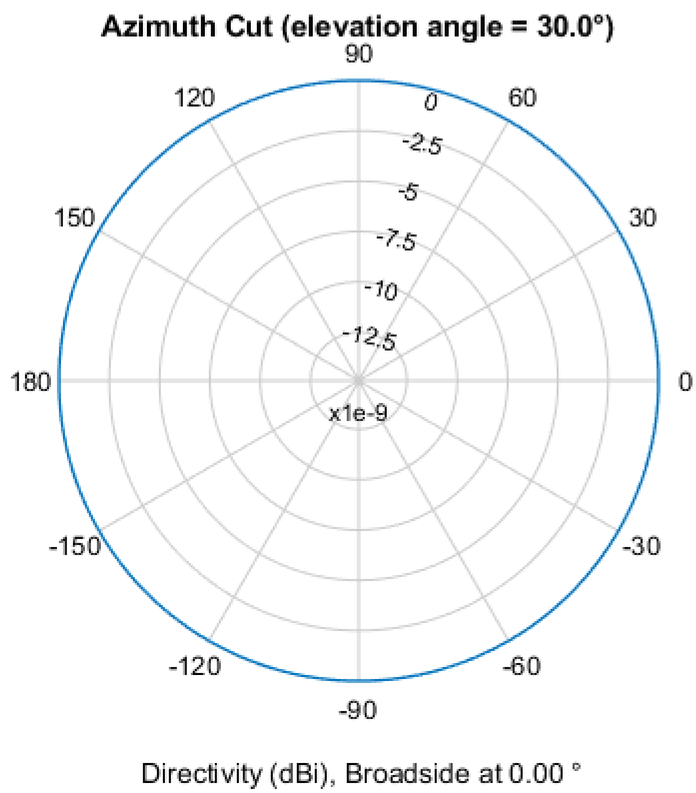
Examples

Azimuth Pattern of Isotropic Projector

Examine the azimuth pattern of an isotropic projector at 30° elevation. The frequency range is between 1 kHz and 10 kHz. Specify the voltage response as a scalar.

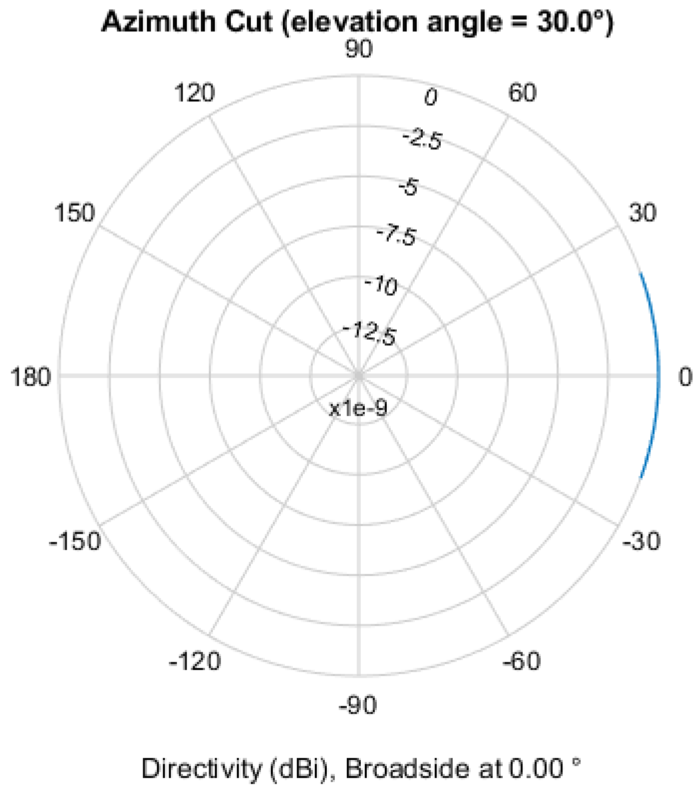
Set the projector parameters.

```
fc = 3e3;
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3, ...
    'VoltageResponse',-115);
patternAzimuth(projector,fc,30)
```



Plot a smaller range of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(projector,fc,30,'Azimuth',[-20:20])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2017a

See Also

`pattern` | `patternElevation`

patternElevation

System object: `phased.IsotropicProjector`
Package: `phased`

Plot isotropic projector directivity and response patterns versus elevation

Syntax

```
patternElevation(projector,FREQ)
patternElevation(projector,FREQ,AZ)
patternElevation(projector,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(projector,FREQ)` plots the 2D element directivity pattern versus elevation (in dBi) for the projector, `projector`, at zero-degrees azimuth angle. The argument `FREQ` specifies the operating frequency.

`patternElevation(projector,FREQ,AZ)`, in addition, plots the 2D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(projector,FREQ,AZ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`projector` — Isotropic projector
`phased.IsotropicProjector` System object

Isotropic projector, specified as a `phased.IsotropicProjector` System object.

Example: `phased.IsotropicProjector`

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

Elevation — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an P-by-N real-valued matrix. The dimension P is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension N is the number of azimuth angles determined by the `AZ` argument.

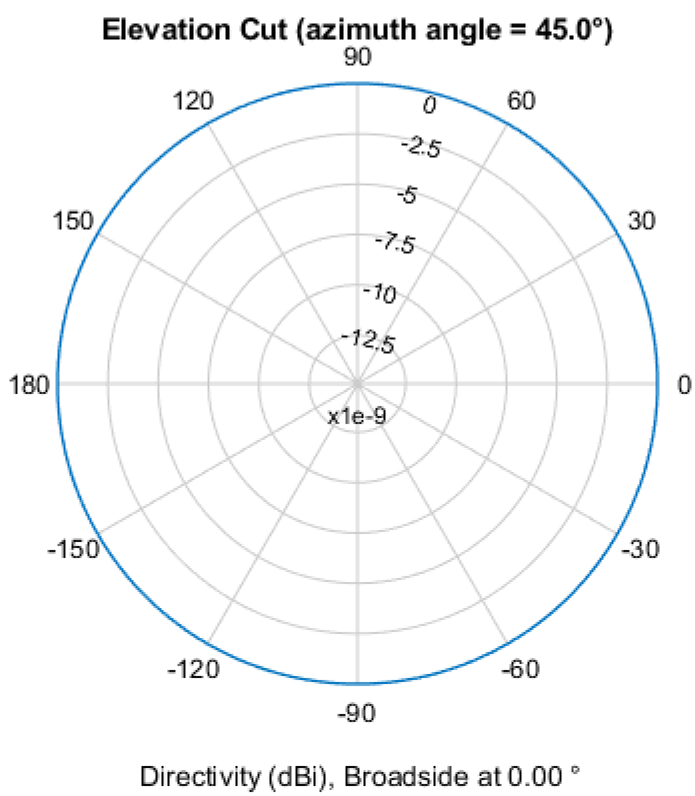
Examples

Elevation Pattern of Isotropic Projector

Plot an elevation cut of the directivity of an isotropic projector at 45° azimuth. Assume the signal frequency is 3 kHz.

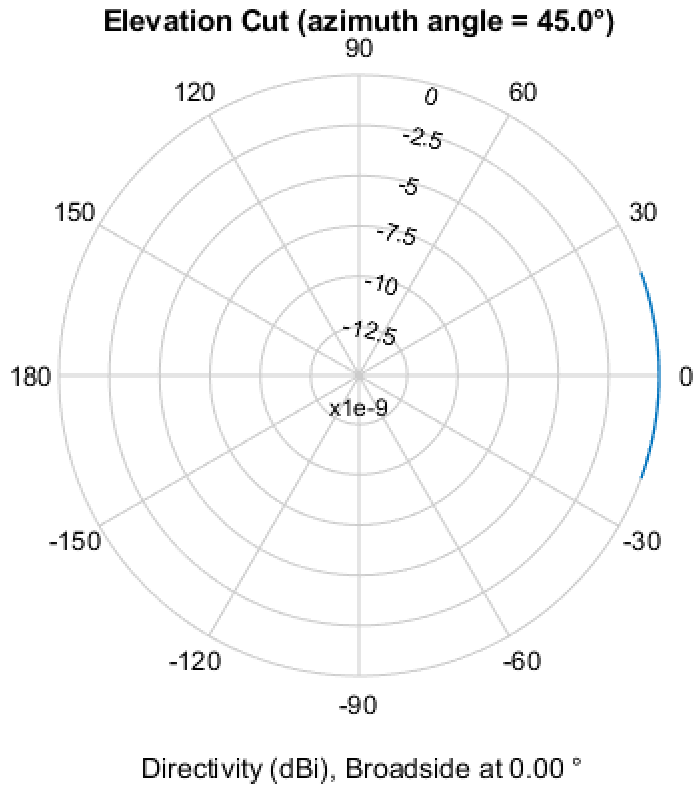
Create the isotropic projector object and call the pattern object function.

```
fc = 3e3;
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3, ...
    'VoltageResponse',70);
patternElevation(projector,fc,45)
```



Plot a smaller range of elevation angles using the `Elevation` parameter.

```
patternElevation(projector,fc,45,'Elevation',-20:20)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2017a

See Also

`pattern` | `patternAzimuth`

step

System object: `phased.IsotropicProjector`
Package: `phased`

Voltage response of isotropic projector

Syntax

```
resp = step(projector,freq,ang)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`resp = step(projector,freq,ang)` returns the voltage response for the projector at the specified operating frequencies and in the specified directions of arriving signals.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`projector` — Isotropic projector
`phased.IsotropicProjector` System object

Isotropic projector, specified as a `phased.IsotropicProjector` System object.

Example: `phased.IsotropicProjector`

`freq` — Voltage response frequencies
positive real scalar | real-valued 1-by-*L* vector of positive values

Voltage response frequencies of projector, specified as a positive real scalar or a real-valued 1-by-*L* vector of positive values. Units are in Hz.

Data Types: `double`

`ang` — Direction of arriving signals
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Direction of arriving signals, specified as a real-valued 1-by-*M* row vector or 2-by-*M* matrix. When `ang` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

When `ang` is a 1-by-*M* row vector, each element specifies the azimuth angle of the arriving signal. In this case, the corresponding elevation angle is assumed to be zero.

Data Types: `double`

Output Arguments

`resp` — Voltage response of projector
real-valued *M*-by-*L* matrix

Voltage response of projector, returned as a real-valued *M*-by-*L* matrix. *M* represents the number of angles specified in `ang`, and *L* represents the number of frequencies specified in `freq`. Units are in V/Pa.

Examples

Response and Pattern of Isotropic Projector at Single Frequency

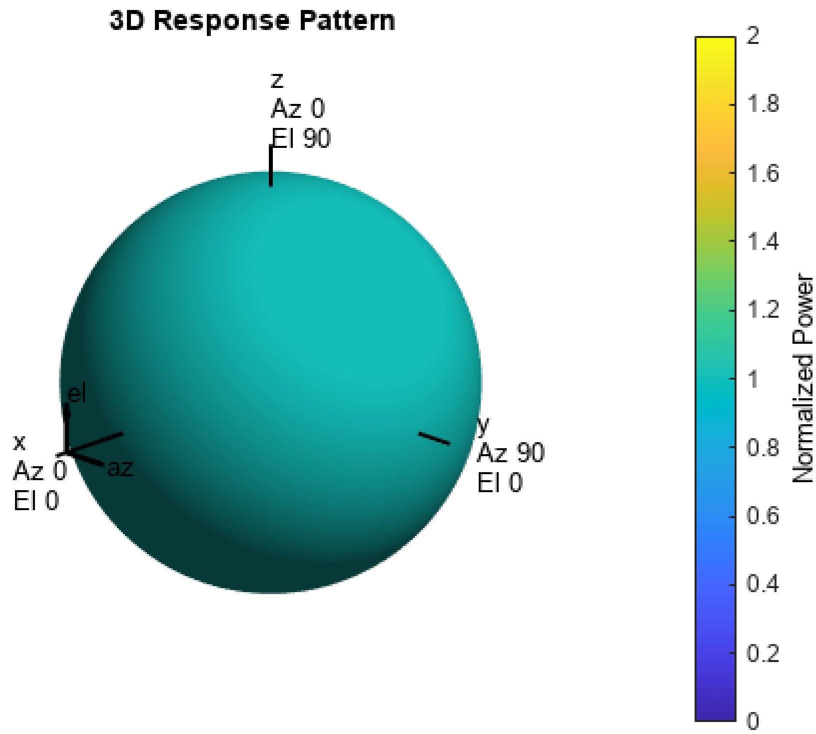
Examine the response and patterns of an isotropic projector operating between 1 kHz and 10 kHz.

Set the projector parameters and obtain the voltage response at five different elevation angles: -30°, -15°, 0°, 15° and 30°. All elevation angles at 0° azimuth angle. The voltage response is computed at 2 kHz.

```
projector = phased.IsotropicProjector('FrequencyRange',[1,10]*1e3);
fc = 2e3;
resp = projector(fc,[0,0,0,0,0;-30,-15,0,15,30]);
```

Draw a 3-D plot of the voltage response.

```
pattern(projector,fc,[-180:180],[-90:90],'CoordinateSystem','polar', ...
    'Type','power')
```



Response and Pattern of Isotropic Projector at Multiple Frequencies

Examine the response and patterns of an isotropic projector at three different frequencies. The projector operates between 1 kHz and 10 kHz. Specify the voltage response as a vector.

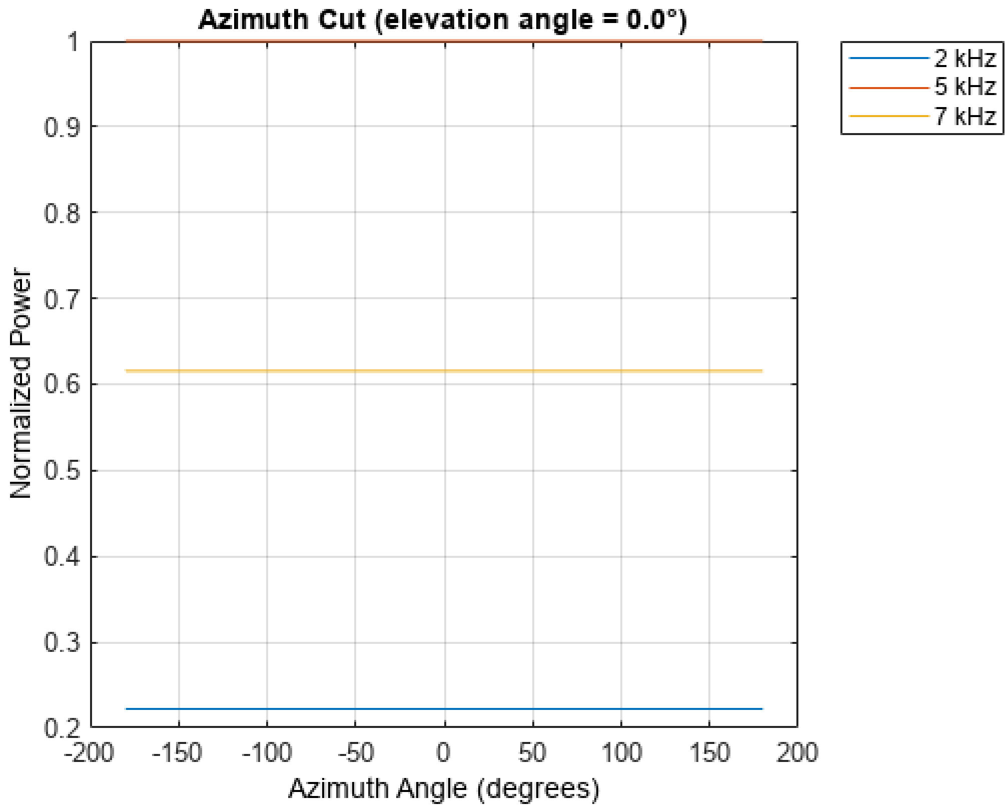
Set up the projector parameters, and obtain the voltage response at 45° azimuth and 30° elevation. Compute the responses at signal frequencies of 2, 5, and 7 kHz.

```
projector = phased.IsotropicProjector('FrequencyRange',[1 10]*1e3, ...
    'VoltageResponse',[90 95 100 95 90]);
fc = [2e3 5e3 7e3];
resp = projector(fc,[45;30]);
resp
```

```
resp = 1×3

    0.0426    0.0903    0.0708
```

Next, draw a 2-D plot of the voltage response as a function of azimuth.

```
pattern(projector,fc,[-180:180],0,'CoordinateSystem','rectangular', ...
    'Type','power')
```



Algorithms

The total response of a projector is a combination of its frequency response and spatial response. `phased.IsotropicProjector` calculates both responses using nearest neighbor interpolation, and then multiplies the responses to form the total response.

See Also

`uv2azel` | `phitheta2azel`

phased.LCMVBeamformer

Package: `phased`

Narrowband LCMV beamformer

Description

The phased.LCMVBeamformer object implements a narrowband linear-constraint minimum-variance (LCMV) beamformer for a sensor array. The LCMV beamformer belongs to the family of constrained optimization beamformers.

To beamform signals arriving at a sensor array:

1. Create the `phased.LCMVBeamformer` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
beamformer = phased.LCMVBeamformer
beamformer = phased.LCMVBeamformer(Name,Value)
```

Description

`beamformer = phased.LCMVBeamformer` creates an LCMV beamformer System object, `beamformer`, with default property values.

`beamformer = phased.LCMVBeamformer(Name,Value)` creates an LCMV beamformer with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `beamformer = phased.LCMVBeamformer('Constraint',[1;1])` sets the constraint matrix.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Constraint` — Constraint matrix
`[1;1]` (default) | complex-valued *N*-by-*K* matrix

Constraint matrix, specified as a complex-valued N-by-K matrix. Each column of the matrix represents a constraint. N is the number of elements in the sensor array and K is the number of constraints. K must be less than or equal to N, $K \leq N$.

Example: `[1 1i;1 1i]`

Data Types: `single` | `double`
Complex Number Support: Yes

`DesiredResponse` — Desired response
`1` (default) | complex-valued K-by-1 vector

Desired response of the LCMV beamformer, specified as a complex-valued K-by-1 vector, where K is the number of constraints in the Constraint property. Each element in the vector defines the desired response of the constraint specified in the corresponding column of the `Constraint` property. A value of one creates a distortionless response and a value of zero creates a null response.

Example: `[1;0]`

Data Types: `single` | `double`
Complex Number Support: Yes

`DiagonalLoadingFactor` — Diagonal loading factor
`0` (default) | nonnegative scalar

Diagonal loading factor, specified as a nonnegative scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample size is small. A small sample size can lead to an inaccurate estimate of the covariance matrix. Diagonal loading also provides robustness due to steering vector errors. The diagonal loading technique adds a positive scalar multiple of the identity matrix to the sample covariance matrix.

Tunable: Yes

Data Types: `single` | `double`

`TrainingInputPort` — Enable training data input
`false` (default) | `true`

Enable training data input, specified as `false` or `true`. When you set this property to `true`, use the training data input argument, `XT`, when running the object. Set this property to `false` to use the input data, `X`, as the training data.

Data Types: `logical`

`WeightsOutputPort` — Enable beamforming weights output
`false` (default) | `true`

Enable the output of beamforming weights, specified as `false` or `true`. To obtain the beamforming weights, set this property to `true` and use the corresponding output argument, `W`. If you do not want to obtain the weights, set this property to `false`.

Data Types: `logical`

Usage

Syntax

```
Y = beamformer(X)
Y = beamformer(X,XT)
[Y,W] = beamformer( ___ )
```

Description

`Y = beamformer(X)` performs LCMV beamforming on the input array data, `X`, and returns the beamformed output in `Y`.

`Y = beamformer(X,XT)` uses `XT` as training data to calculate the beamforming weights. To use this syntax, set the TrainingInputPort property to `true`.

`[Y,W] = beamformer( ___ )` returns the beamforming weights `W`. To use this syntax, set the WeightsOutputPort property to `true`.

Input Arguments

X — Array element data
complex-valued *M*-by-*N* matrix

Array element data, specified as an*M*-by-*N* matrix where *N* is the number of elements in the sensor array.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1 0.5 2.6; 2 -0.2 0]`

Data Types: `single` | `double`

XT — Training data
complex-valued *P*-by-*N* matrix

Training data, specified as a *P*-by-*N* matrix. *N* is the number of elements of the sensor array. *P* is the length of the training data and must be greater than *N*.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1 0.5 2.6; 2 -0.2 0; 3 -2 -1]`

Dependencies

To enable this argument, set the `TrainingInputPort` property to `true`.

Data Types: `single` | `double`

Output Arguments

Y — Beamformed output
complex-valued *M*-by-1 vector

Beamformed output, returned as a complex-valued *M*-by-1 vector.

Data Types: `single` | `double`

W — Beamformer weights
complex-valued *N*-by-1 vector

Beamformer weights, returned as a complex-valued *N*-by-1 vector. *N* is the number of elements in the sensor array.

Dependencies

To enable this argument, set the WeightsOutputPort property to `true`.

Data Types: `single` | `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

LCMV Beamformer with One Constraint

Apply an LCMV beamformer to a 5-element ULA of isotropic sensor elements, preserving the signal from a desired direction. The operating frequency is 300 MHz.

Simulate a low-frequency sinusoid signal in Gaussian noise.

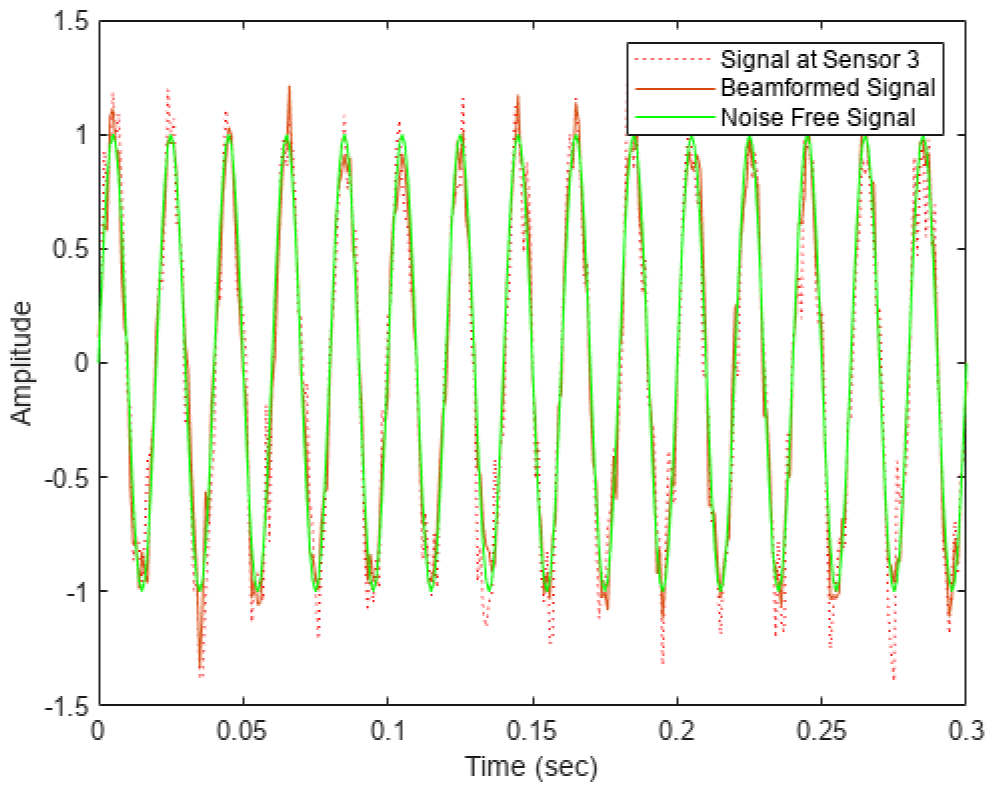
```
f = 50;
t = (0:.001:.3)';
x = sin(2*pi*f*t);
c = physconst('LightSpeed');
fc = 300e6;
lambda = c/fc;
incidentAngle = [45;0];
antenna = phased.IsotropicAntennaElement('FrequencyRange',[20 20e8]);
array = phased.ULA('NumElements',5,'ElementSpacing',lambda/2,...
    'Element',antenna);
x = collectPlaneWave(array,x,incidentAngle,fc,c);
noise = 0.2*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;
```

Beamform the array.

```
steervec = phased.SteeringVector('SensorArray',array,...
    'PropagationSpeed',c);
beamformer = phased.LCMVBeamformer('Constraint',steervec(fc,incidentAngle),'DesiredResponse',1);
y = beamformer(rx);
```

Plot the original and beamformed signals.

```
plot(t,real(rx(:,3)),'r:',t,real(y),t,real(x(:,3)),'g')
xlabel('Time (sec)')
ylabel('Amplitude')
legend('Signal at Sensor 3','Beamformed Signal','Noise Free Signal')
```



Nulling with LCMV Beamformer

This example shows how to use an LCMV beamformer to point a null of the array response in the direction of an interfering source. The array is a 10-element uniform linear array (ULA). By default, the ULA elements are isotropic antennas created by the `phased.IsotropicAntennaElement` System object™. Set the frequency range of the antenna elements so that the carrier frequency lies within the operating range. The carrier frequency is 1 GHz.

```
fc = 1e9;
lambda = physconst('LightSpeed')/fc;
array = phased.ULA('NumElements',10,'ElementSpacing',lambda/2);
array.Element.FrequencyRange = [8e8 1.2e9];
```

Simulate a test signal using a simple rectangular pulse.

```
t = linspace(0,0.3,300)';
testsig = zeros(size(t));
testsig(201:205) = 1;
```

Assume the rectangular pulse is incident on the ULA from an angle of 30° azimuth and 0° elevation. Use the `collectPlaneWave` function of the ULA System object to simulate reception of the pulse waveform from the incident angle.

```
angle_of_arrival = [30;0];
x = collectPlaneWave(array,testsig,angle_of_arrival,fc);
```

The signal `x` is a matrix with ten columns. Each column represents the received signal at one of the array elements.

Construct a conventional phase-shift beamformer. Set the `WeightsOutputPort` property to `true` to output the spatial filter weights.

```
convbeamformer = phased.PhaseShiftBeamformer('SensorArray',array,...
    'OperatingFrequency',1e9,'Direction',angle_of_arrival,...
    'WeightsOutputPort',true);
```

Add complex-valued white Gaussian noise to the signal `x`. Set the default random number stream for reproducible results.

```
rng default
npower = 0.5;
x = x + sqrt(npower/2)*(randn(size(x)) + 1i*randn(size(x)));
```

Create a 10W interference source. Specify the barrage jammer to have an effective radiated power of 10 W. The interference signal from the barrage jammer is incident on the ULA from an angle of 120° azimuth and 0° elevation. Use the `collectPlaneWave` function of the ULA System object to simulate reception of the jammer signal.

```
jamsig = sqrt(10)*randn(300,1);
jammer_angle = [120;0];
jamsig = collectPlaneWave(array,jamsig,jammer_angle,fc);
```

Add complex-valued white Gaussian noise to simulate noise contributions not directly associated with the jamming signal. Again, set the default random number stream for reproducible results. This noise power is 0 dB below the jammer power. Beamform the signal using a conventional beamformer.

```
noisePwr = 1e-5;
rng(2008);
noise = sqrt(noisePwr/2)*...
    (randn(size(jamsig)) + 1j*randn(size(jamsig)));
jamsig = jamsig + noise;
rxsig = x + jamsig;
[yout,w] = convbeamformer(rxsig);
```

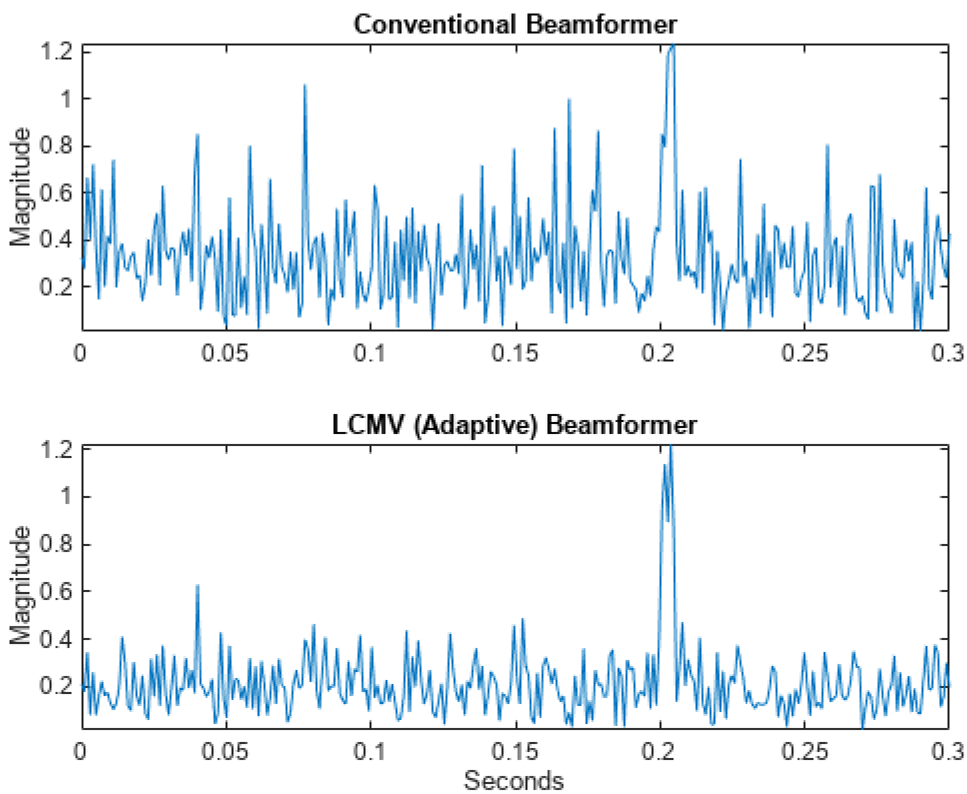
Implement the adaptive LCMV beamformer using the same ULA array. Use the target-free data, `jamsig`, as training data. Output the beamformed signal and the beamformer weights.

```
steeringvector = phased.SteeringVector('SensorArray',array,...
    'PropagationSpeed',physconst('LightSpeed'));
LCMVbeamformer = phased.LCMVBeamformer('DesiredResponse',1,...
    'TrainingInputPort',true,'WeightsOutputPort',true);
```

```
LCMVbeamformer.Constraint = steeringvector(fc,angle_of_arrival);
LCMVbeamformer.DesiredResponse = 1;
[yLCMV,wLCMV] = LCMVbeamformer(rxsig,jamsig);
```

Plot the conventional beamformer output and the adaptive beamformer output.

```
subplot(211)
plot(t,abs(yout))
axis tight
title('Conventional Beamformer')
ylabel('Magnitude')
subplot(212)
plot(t,abs(yLCMV))
axis tight
title('LCMV (Adaptive) Beamformer')
xlabel('Seconds')
ylabel('Magnitude')
```

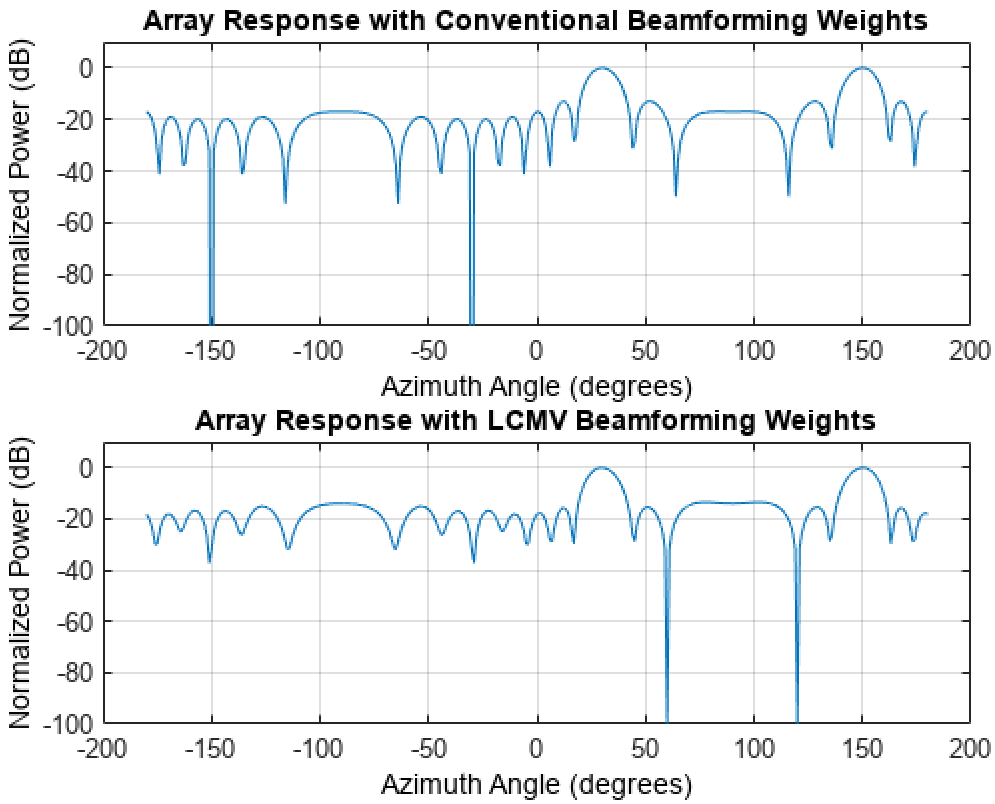


The adaptive beamformer significantly improves the SNR of the rectangular pulse at 0.2 s.

Using conventional and LCMV weights, plot the responses for each beamformer.

```
subplot(211)
pattern(array,fc,[-180:180],0,'PropagationSpeed',physconst('LightSpeed'),...
    'CoordinateSystem','rectangular','Type','powerdb','Normalize',true,...
    'Weights',w)
title('Array Response with Conventional Beamforming Weights');
subplot(212)
```

```
pattern(array,fc,[-180:180],0,'PropagationSpeed',physconst('LightSpeed'),...)
    'CoordinateSystem','rectangular','Type','powerdb','Normalize',true,...
    'Weights',wLCMV)
title('Array Response with LCMV Beamforming Weights');
```



The adaptive beamform places a null at the arrival angle of the interference signal, 120°.

Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also
`phased.MVDRBeamformer` | `phased.PhaseShiftBeamformer` | `phased.TimeDelayLCMVBeamformer`

Topics
"Adaptive Beamforming"

step

System object: `phased.LCMVBeamformer`
Package: `phased`

Perform LCMV beamforming

Syntax

```
Y = step(H,X)
Y = step(H,X,XT)
[Y,W] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` performs LCMV beamforming on the input, `X`, and returns the beamformed output in `Y`. `X` is an M-by-N matrix where N is the number of elements of the sensor array. `Y` is a column vector of length M.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`Y = step(H,X,XT)` uses `XT` as the training samples to calculate the beamforming weights. This syntax is available when you set the `TrainingInputPort` property to `true`. `XT` is a P-by-N matrix, where N is the number of elements of the sensor array. P must be greater than N.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`[Y,W] = step( ___ )` returns the beamforming weights `W`. This syntax is available when you set the `WeightsOutputPort` property to `true`. `W` is a column vector of length N, where N is the number of elements in the sensor array.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

LCMV Beamformer with One Constraint

Apply an LCMV beamformer to a 5-element ULA of isotropic sensor elements, preserving the signal from a desired direction. The operating frequency is 300 MHz.

Simulate a low-frequency sinusoid signal in Gaussian noise.

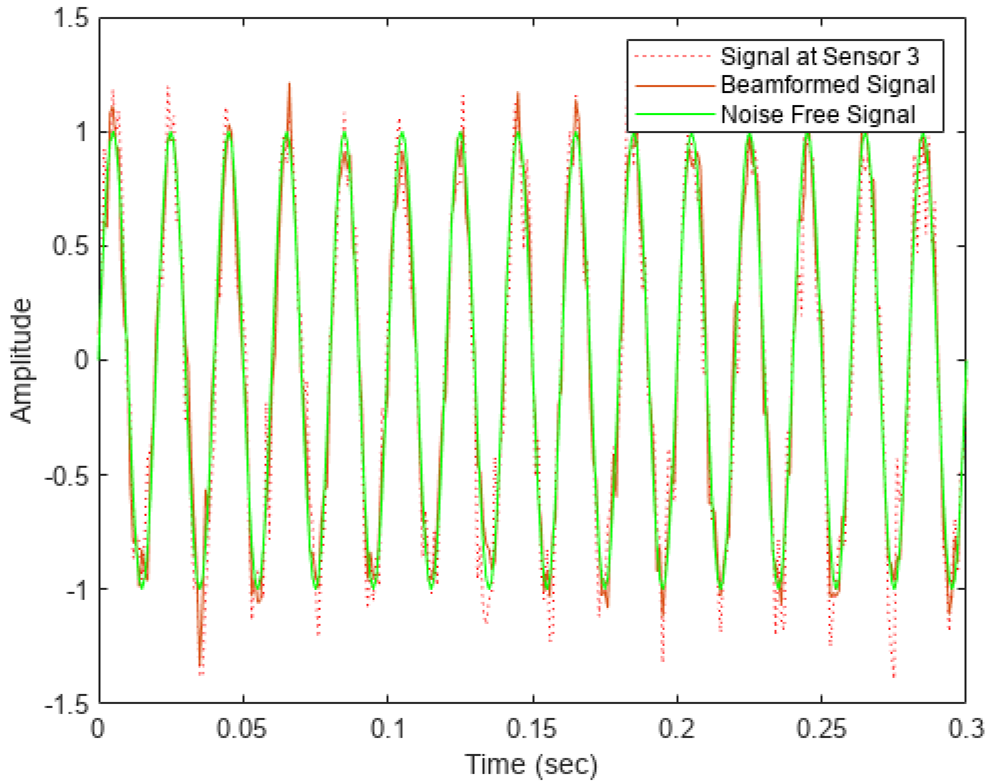
```
f = 50;
t = (0:.001:.3)';
x = sin(2*pi*f*t);
c = physconst('LightSpeed');
fc = 300e6;
lambda = c/fc;
incidentAngle = [45;0];
antenna = phased.IsotropicAntennaElement('FrequencyRange',[20 20e8]);
array = phased.ULA('NumElements',5,'ElementSpacing',lambda/2,...
    'Element',antenna);
x = collectPlaneWave(array,x,incidentAngle,fc,c);
noise = 0.2*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;
```

Beamform the array.

```
steervec = phased.SteeringVector('SensorArray',array,...
    'PropagationSpeed',c);
beamformer = phased.LCMVBeamformer('Constraint',steervec(fc,incidentAngle),'DesiredResponse',1);
y = beamformer(rx);
```

Plot the original and beamformed signals.

```
plot(t,real(rx(:,3)),'r:',t,real(y),t,real(x(:,3)),'g')
xlabel('Time (sec)')
ylabel('Amplitude')
legend('Signal at Sensor 3','Beamformed Signal','Noise Free Signal')
```



Signal at Sensor 3
Beamformed Signal
Noise Free Signal
Amplitude
Time (sec)
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.05
0.1
0.15
0.2
0.25
0.3

phased.LinearFMWaveform

Package: `phased`

Linear FM pulse waveform

Description

The `LinearFMWaveform` System object creates a linear FM pulse waveform.

To create the waveform:

1. Create the `phased.LinearFMWaveform` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
waveform = phased.LinearFMWaveform
waveform = phased.LinearFMWaveform(Name=Value)
```

Description

`waveform = phased.LinearFMWaveform` creates a linear FM pulse `waveform` System object. Use this object to generate samples of a linear FM pulse waveform.

`waveform = phased.LinearFMWaveform(Name=Value)` creates a linear FM pulse `waveform` System object with each specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1=Value1`,...,`NameN=ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SampleRate` — Signal sample rate
`1e6` (default) | positive scalar

Signal sample rate, specified as a positive scalar. The ratio of sample rate to pulse repetition frequency must be a positive integer, so the number of samples in each pulse must be an integer value. Units are in Hertz.

Example: 100e3

Data Types: `double`

`DurationSpecification` — Method to set pulse duration
`'Pulse width'` (default) | `'Duty cycle'`

Method to set pulse duration (pulse width), specified as `'Pulse width'` or `'Duty cycle'`. This property determines how you set the pulse duration.

- When you set this property to `'Pulse width'`, set the pulse duration directly using the `PulseWidth` property.
- When you set this property to `'Duty cycle'`, set the pulse duration from the values of the `PRF` and `DutyCycle` properties. The pulse width is equal to the value of the `DutyCycle` property divided by the value of the `PRF` property.

Data Types: `char` | `string`

`PulseWidth` — Pulse time duration
`50e-6` (default) | positive scalar

Pulse time duration, specified as a positive scalar. The value must satisfy `PulseWidth <= 1./PRF`. Units are in seconds.

Dependencies

To enable this property, set the `DurationSpecification` property to `'Pulse width'`.

Data Types: `double`

`DutyCycle` — Pulse duty cycle
`0.5` (default) | positive scalar in the range `[0,1]`

Pulse duty cycle, specified as a positive scalar in the range `[0,1]`. The pulse width is the value of the `DutyCycle` property divided by the value of the `PRF` property. This quantity is dimensionless.

Example: `0.75`

Dependencies

To enable this property, set the `DurationSpecification` property to `'Duty cycle'`.

Data Types: `double`

`PRF` — Pulse repetition frequency
`10e3` (default) | positive scalar | row vector of positive values

Pulse repetition frequency (PRF), specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval (PRI) is the inverse of the pulse repetition frequency PRF value. The PRF must satisfy these restrictions:

- The product of `PRF` and `PulseWidth` must be less than or equal to one. This condition requires that the pulse width is less than one PRI. For the phase-coded waveform, the pulse width is the product of the values of the `ChipWidth` and `NumChips` properties.
- The ratio of `SampleRate` to `PRF` must be an integer. This condition requires that the number of samples in one PRI is an integer.

You can set the value of PRF using the PRF property settings alone or using property settings in conjunction with the prfidx input argument of the object.

- When PRFSelectionInputPort is false, you set the PRF using the PRF properties alone. You can:
  - Implement a constant PRF by specifying the PRF property as a positive real-valued scalar.
  - Implement a staggered PRF by specifying the PRF property as a row vector with positive real-valued elements. Each call to the object uses successive elements of this vector as the PRF. Once the object reaches the last element of the vector, it continues the process cyclically with the first element of the vector.
- When PRFSelectionInputPort is true, you can set the PRF value using the PRF property in conjunction with the prfidx input argument. You implement a selectable PRF by specifying the PRF property as a row vector with positive real-valued elements. When you execute the object, the object selects a PRF by using the index you specify in the prfidx input argument to index into the PRF vector.

In all cases, the number of output samples is fixed when you set the OutputFormat property to 'Samples'. When you use a varying PRF and also set the OutputFormat property to 'Pulses', the number of samples can vary.

Data Types: double

PRFSelectionInputPort — Enable PRF selection input
false (default) | true

Enable PRF selection input, specified as false or true. When you set this property to true, you can pass an index argument to the object to select a predefined value from the PRF property vector. When you set this property to false, the object uses the PRF property to define the PRF sequence used in the simulation.

Data Types: logical

SweepBandwidth — Bandwidth of linear FM sweep
100e3 (default) | positive scalar

Bandwidth of linear FM sweep, specified as a positive scalar. Units are in Hz.

Data Types: double

SweepDirection — Direction of linear FM sweep
'Up' (default) | 'Down'

Direction of the linear FM sweep, specified as 'Up' or 'Down'.

Data Types: char | string

SweepInterval — Location of FM sweep interval
'Positive' (default) | 'Symmetric'

Location of the FM sweep interval, specified as'Positive' or 'Symmetric'.

- If SweepInterval is 'Positive', the waveform sweeps the interval between 0 and B where B is the sweep bandwidth in the SweepBandwidth property.
- If SweepInterval is 'Symmetric', the waveform sweeps the interval between $-B/2$ and $B/2$.

Data Types: `char` | `string`

`Envelope` — Waveform envelope function
`'Rectangular'` (default) | `'Gaussian'`

Waveform envelope function, specified as `'Rectangular'` or `'Gaussian'`.

Data Types: `char` | `string`

`FrequencyOffsetSource` — Source of frequency offset
`'Property'` (default) | `'Input port'`

Source of frequency offset, specified as `'Property'` or `'Input port'`.

- When you set this property to `'Property'`, the frequency offset is determined by the value of the `FrequencyOffset` property.
- When you set this property to `'Input port'`, the frequency offset is determined by the input argument `freqoffset` when calling the object.

Example: `'Input port'`

Data Types: `char` | `string`

`FrequencyOffset` — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `150.0`

Dependencies

To enable this property, set the `FrequencyOffsetSource` property to `'Property'`.

Data Types: `double`

`OutputFormat` — Format of output signal
`'Pulses'` (default) | `'Samples'`

Format of output signal, specified as `'Pulses'` or `'Samples'`.

- When you set the `OutputFormat` property to `'Pulses'`, the output of the object takes the form of multiple pulses specified by the value of the `NumPulses` property. The number of samples per pulse can vary if you change the PRF during the simulation.
- When you set the `OutputFormat` property to `'Samples'`, the output of the object takes the form of multiple samples. In this case, the number of output signal samples is the value of the `NumSamples` property and is fixed.

Data Types: `char` | `string`

`NumSamples` — Number of samples in output
`100` (default) | positive integer

Number of samples in each output of the object, specified as a positive integer.

Dependencies

To enable this property, set the `OutputFormat` property to `'Samples'`.

Data Types: `double`

`NumPulses` — Number of pulses in output
`1` (default) | positive integer

Number of pulses in each output, specified as a positive integer.

Dependencies

To enable this property, set the `OutputFormat` property to `'Pulses'`.

Data Types: `double`

`PRFOutputPort` — Enable PRF output
`false` (default) | `true`

Enable PRF output, specified as `false` or `true`. Set this property to `true` to output the PRF.

Dependencies

To enable this property, set the `OutputFormat` property to `'Pulses'`.

Data Types: `logical`

`CoefficientsOutputPort` — Enable matched filter coefficients output
`false` (default) | `true`

Enable matched filter coefficients output, specified as `false` or `true`. Set this property to `true` to enable the object the output of the matched filter coefficients of the waveform used during the simulation.

Data Types: `logical`

Usage

Syntax

```
Y = waveform()
Y = waveform(prfidx)
Y = waveform(freqoffset)
[Y,PRF] = waveform( ___ )
[Y,coeff] = waveform( ___ )
```

Description

`Y = waveform()` returns samples of the linear FM pulse in a column vector `Y`. `Y` can contain either a certain number of pulses or a certain number of samples.

`Y = waveform(prfidx)`, specifies the index `prfidx` of the pulse repetition frequency (PRF). The index identifies selected entries in the `PRF` property. This syntax applies when you set the `PRFSelectionInputPort` property to `true`.

Use this syntax for the cases where the transmitted pulse needs to be dynamically selected. In such situations, the `PRF` property includes a list of predetermined choices of PRF's. Based on `prfidx` value input, one of the PRF's is selected as the PRF for the next transmission.

Note that the transmission always finishes the current pulse before starting the next pulse. Therefore, when you set the `OutputFormat` property to `'Samples'` and then specify the `NumSamples` property to be shorter than a pulse, it is possible that during a given simulation step, if the entire output is needed to finish the previously transmitted pulse, the specified `prfidx` is ignored.

`Y = waveform(freqoffset)` generates a waveform with a frequency offset `freqoffset`. Use this syntax for cases where the transmit pulse frequency needs to be dynamically updated.

This syntax applies when you set the `FrequencyOffsetSource` property to `'Input port'`.

`[Y,PRF] = waveform( ___ )` also returns the current pulse repetition frequency `PRF`. To enable this syntax, set the `PRFOutputPort` property to `true` and set the `OutputFormat` property to `'Pulses'`.

`[Y,coeff] = waveform( ___ )` also returns the matched filter coefficients, `coeff`, for the current pulse. To enable this syntax, set the `CoefficientsOutputPort` property to `true`.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example,

```
[Y,PRF,coeff] = waveform(prfidx,freqoffset)
```

Input Arguments

`prfidx` — index of pulse repetition frequency
positive integer

Index of pulse frequency, specified as a positive integer. The index identifies the entries in the `PRF`.

Dependencies

To enable this argument, set the `PRFSelectionInputPort` property to `true`

`freqoffset` — frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. The offset generates the waveform with a frequency offset. Use this argument for the cases where the transmit pulse frequency needs to be dynamically updated. Units are in Hz.

Dependencies

To enable this argument, set the `FrequencyOffsetSource` property to `'Input port'`.

Data Types: `double`

Output Arguments

`PRF` — pulse repetition frequency
scalar

Pulse repetition frequency, returned as a scalar.

Dependencies

To enable this argument, set the `PRFOutputPort` property to `true` and set the `OutputFormat` to `'Pulses'`. When you set the `PRFOutputPort` property to `true`, the function returns the current PRF used by the system.

`coeff` — matched filter coefficients
N_Z-by-1 complex-valued vector | N_Z-by-M complex-valued matrix

Matched filter coefficients, returned as an N_Z-by-1 complex-valued vector or an N_Z-by-M complex-valued matrix.

- If you set `OutputFormat` to `'Pulses'` and `NumPulses` is 1, the object returns `coeff` as an N_Z-length vector. N_Z corresponds to the pulse width.
- If you set `OutputFormat` to `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is `'Samples'` and `DurationSpecification` is `'Pulse width'`, `coeff` is returned as an N_Z-length vector. N_Z corresponds to the pulse width.
- If `OutputFormat` is set to `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is set to `'Samples'` and `DurationSpecification` is set to `'Duty cycle'` with only one unique PRF value, `coeff` is returned as an N_Z-length vector. N_Z corresponds to the pulse width.
- If `OutputFormat` is `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is `'Samples'` and `DurationSpecification` is `'Duty cycle'`, `coeff` is returned as an N_Z-by-M matrix. N_Z corresponds to the maximum pulse width and M corresponds to the number of unique PRFs.

Dependencies

To enable this argument, set the `CoefficientsOutputPort` property to `true`.

Data Types: `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to waveform System objects

bandwidth	Waveform bandwidth
getMatchedFilter	Matched filter coefficients derived from waveform
getStretchProcessor	Create stretch processor for waveform
plot	Plot pulse waveform

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

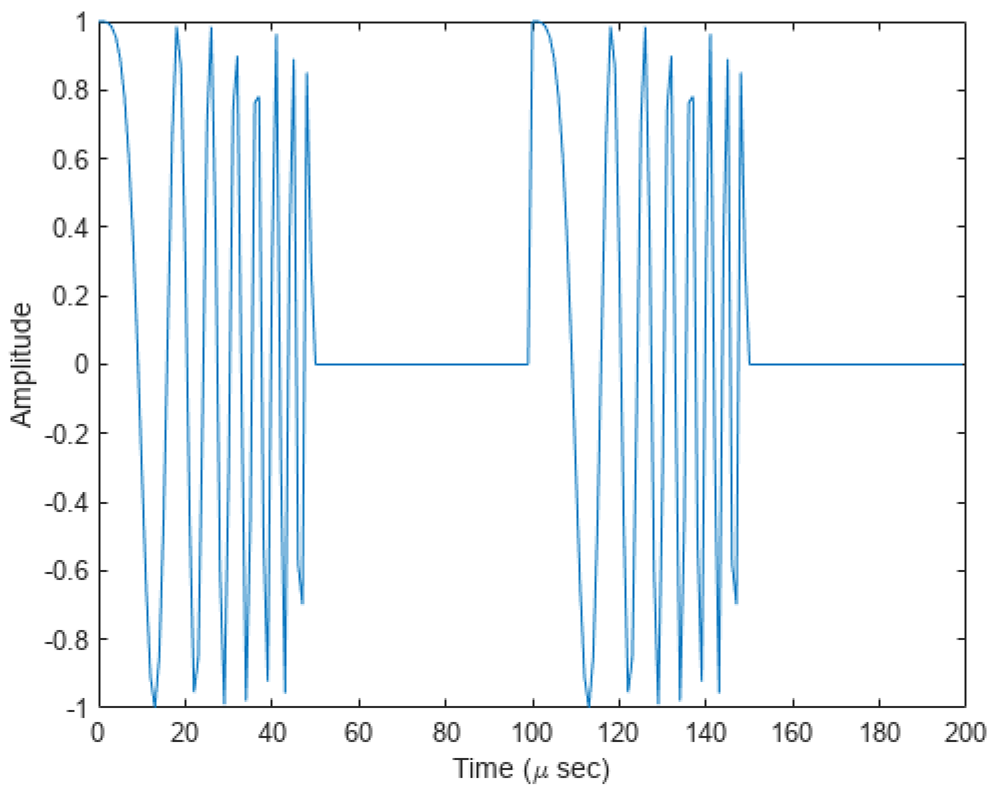
Create Linear FM Pulses

Construct a linear FM waveform having a sweep bandwidth of 300 kHz, a sample rate of 1 MHz, a pulse width of 50 microseconds, and a pulse repetition frequency of 10 kHz. Generate two pulses.

```
sLFM = phased.LinearFMWaveform('SweepBandwidth',3e5,...
    'OutputFormat','Pulses','SampleRate',1e6,...
    'PulseWidth',50e-6,'PRF',10e3,'NumPulses',2);
```

Obtain and plot the linear FM waveform.

```
wav = step(sLFM);
numpulses = size(wav,1);
t = [0:(numpulses-1)]/sLFM.SampleRate;
plot(t*1e6,real(wav))
xlabel('Time (\mu sec)')
ylabel('Amplitude')
```



Create Linear FM Pulses with Variable PRF

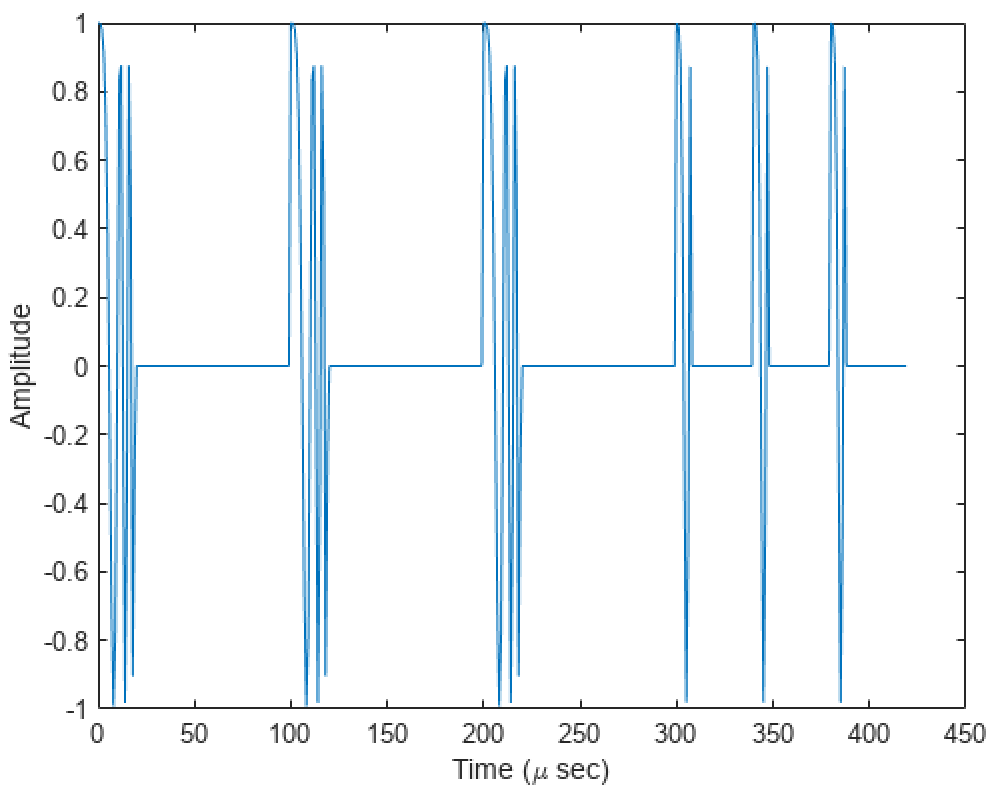
Construct six linear FM waveform pulses having a sweep bandwidth of 300 kHz, a sample rate of 1 MHz, a pulse width of 50 microseconds, and a duty cycle of 20%. Vary the pulse repetition frequency.

Set the sample rate and PRF. The ratio of sample rate to PRF must be an integer.

```
fs = 1e6;
PRF = [10000,25000];
sLFM = phased.LinearFMWaveform('SweepBandwidth',3e5,...
    'OutputFormat','Pulses','SampleRate',fs,...
    'DurationSpecification','Duty Cycle','DutyCycle',.2,...
    'PRF',PRF,'NumPulses',1,'PRFSelectionInputPort',true);
```

Obtain and plot the linear FM waveforms. For the first three calls to the step method, set the PRF to 10 kHz using the PRF index. For the next three calls, set the PRF to 25 kHz.

```
wav = [];
for n = 1:6
    idx = floor((n-1)/3)+1;
    wav1 = step(sLFM,idx);
    wav = [wav;wav1];
end
nsamps = size(wav,1);
t = [0:(nsamps-1)]/sLFM.SampleRate;
plot(t*1e6,real(wav))
xlabel('Time (\mu sec)')
ylabel('Amplitude')
```



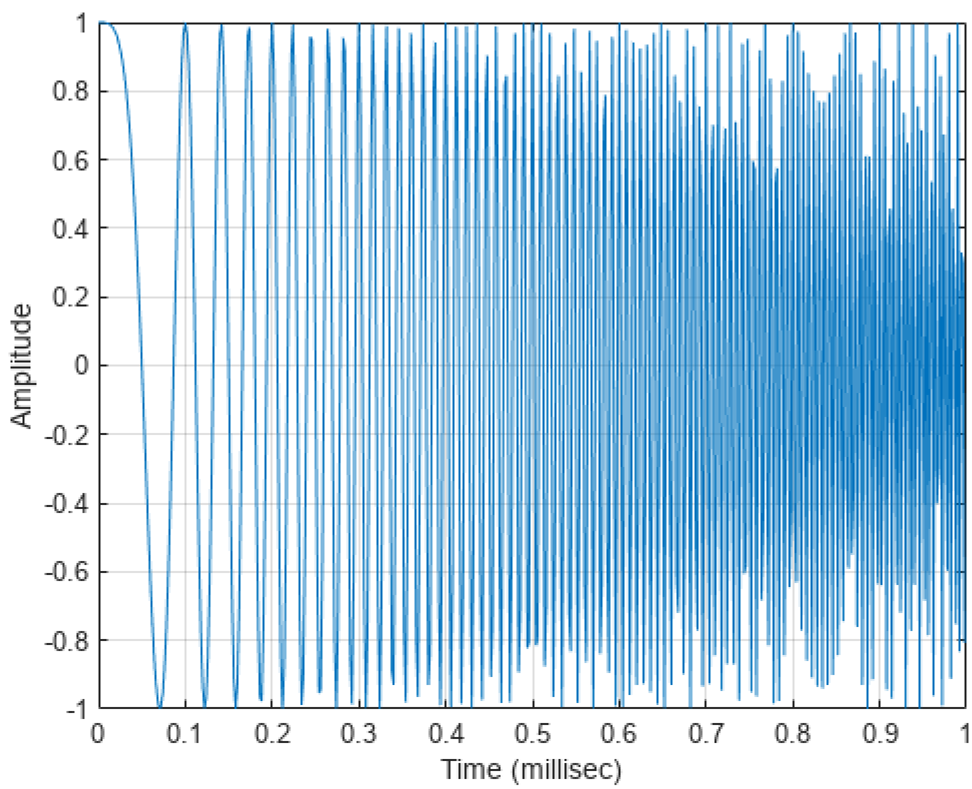
Plot LFM Waveform and Spectrum

Create and plot an upsweep linear FM pulse waveform. The sample rate is 500 kHz, the sweep bandwidth is 200 kHz and the pulse width is 1 millisecond (equal to the pulse repetition interval).

```
fs = 500e3;
sLFM = phased.LinearFMWaveform('SampleRate',fs,...
    'SweepBandwidth',200e3,...
    'PulseWidth',1e-3,'PRF',1e3);
```

Obtain and then plot the real part of the LFM waveform.

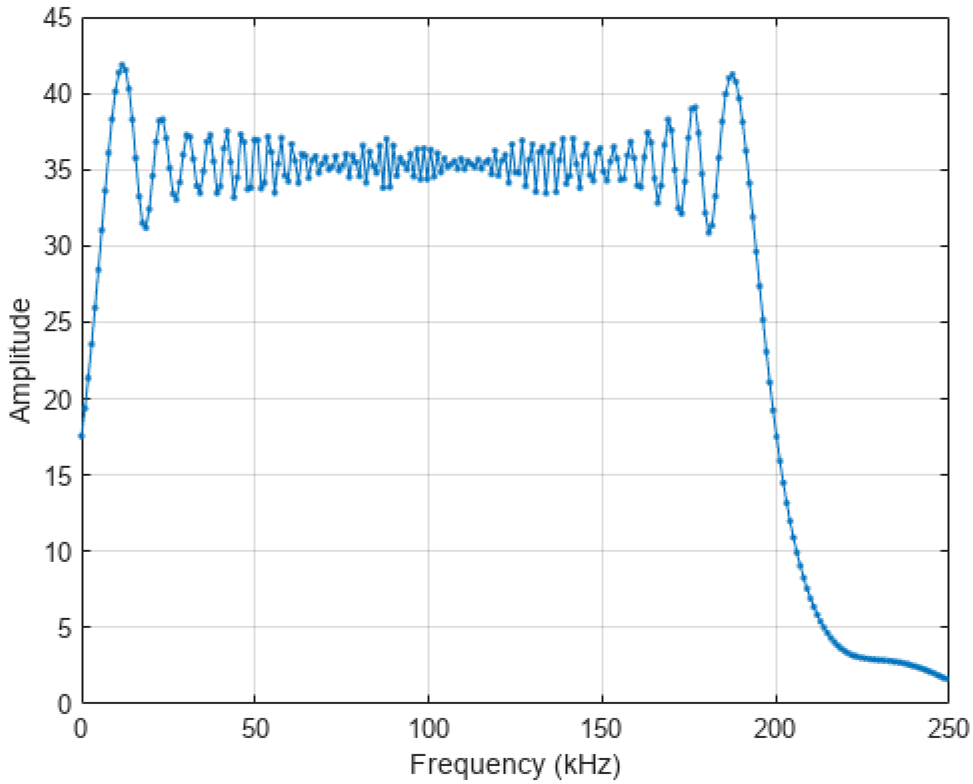
```
lfmwav = step(sLFM);
nsamp = size(lfmwav,1);
t = [0:(nsamp-1)]/fs;
plot(t*1000,real(lfmwav))
xlabel('Time (millisec)')
ylabel('Amplitude')
grid
```



Plot the Fourier transform of the complex signal.

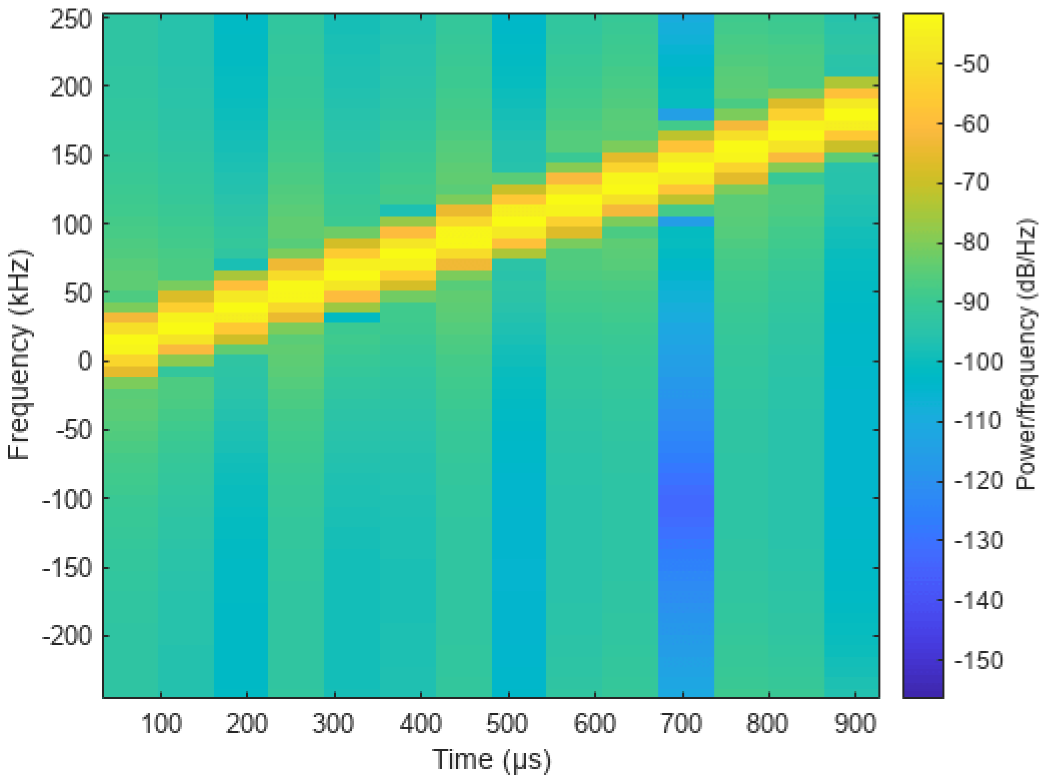
```
nfft = 2^nextpow2(nsamp);
Z = fft(lfmwav,nfft);
fr = [0:(nfft/2-1)]/nfft*fs;
plot(fr/1000,abs(Z(1:nfft/2)),'.-')
xlabel('Frequency (kHz)')
```

```
ylabel('Amplitude')
grid
```



Plot a spectrogram of the function with window size of 64 samples and 50% overlap.

```
nfft1 = 64;
nov = floor(0.5*nfft1);
spectrogram(lfmwav,hamming(nfft1),nov,nfft1,fs,'centered','yaxis')
```



This plot shows the increasing frequency of the signal.

Apply Frequency Offset to Linear FM Waveform

Apply a frequency offset to an upsweep linear FM (LFM) pulse waveform. Plot the frequency spectrum of the waveform with and without applying a frequency offset.

Create an LFM waveform object, which is configured to set the frequency offset from an input when the object is executed.

```
fs = 500e3;
sLFM = phased.LinearFMWaveform('SampleRate',fs,'SweepBandwidth',200e3, ...
    'PulseWidth',2e-5,'PRF',1e3,'FrequencyOffsetSource','Input port');
```

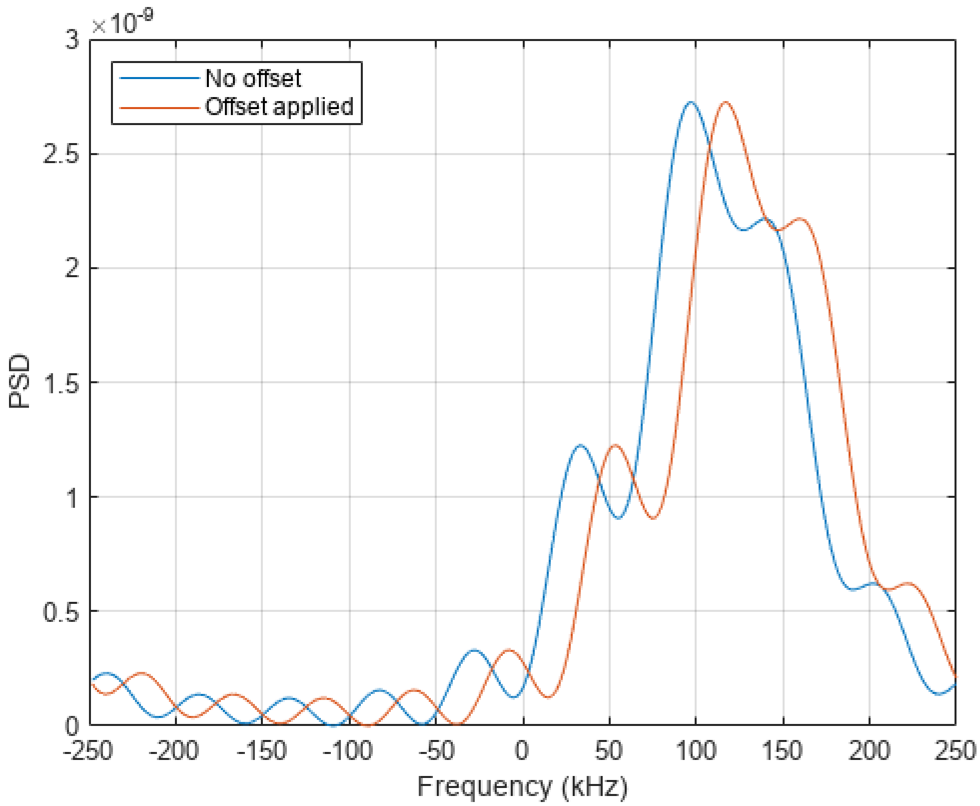
Execute the object two times. First set the frequency offset to 0 Hz and then to 2e4 Hz.

```
lfmwav = sLFM(0);
lfmwav_foffset = sLFM(2e4);
```

Plot the frequency spectrum of the complex signals. The frequency offset signal is shifted to the right.

```
[Pxx,f] = pwelch(lfmwav,[],[],[],fs,'centered');
[Pxx_offset,foffset] = pwelch(lfmwav_foffset,[],[],[],fs,'centered');
plot(f/1000,Pxx,foffset/1000,Pxx_offset)
ylabel('PSD');
```

```
xlabel('Frequency (kHz)');
legend({'No offset','Offset applied'},'Location','northwest');
grid on;
```



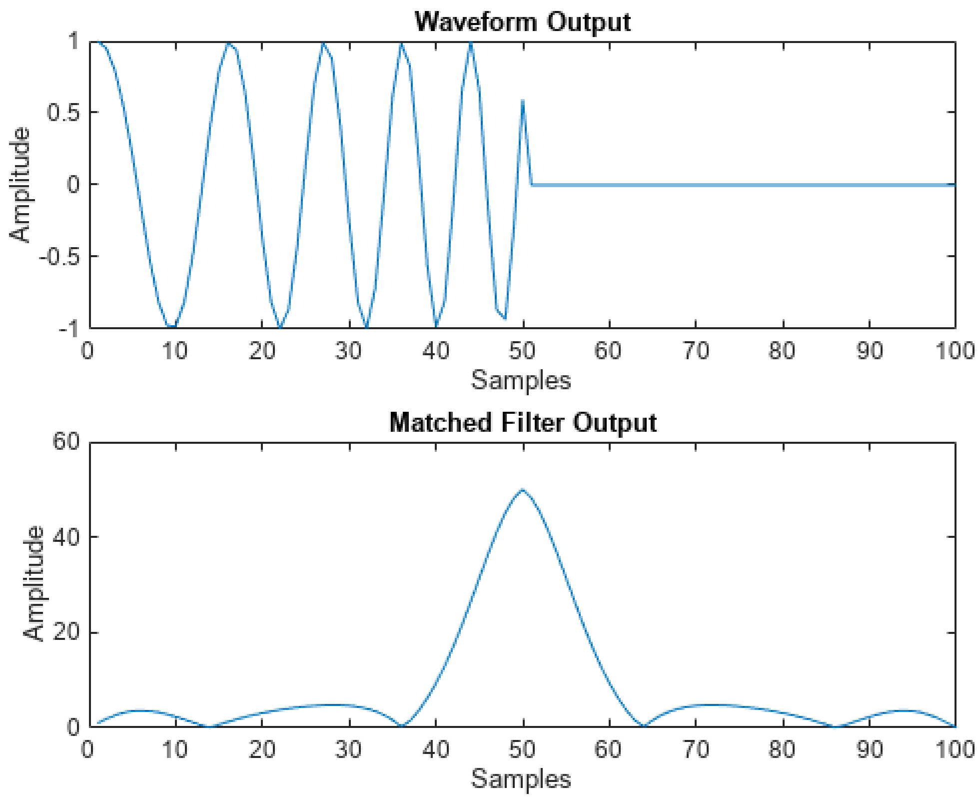
Generate Matched Filter Coefficients of Linear FM Pulse Waveform

Generate output samples and matched filter coefficients of a linear FM pulse waveform at a 50 kHz frequency offset.

```
waveform = phased.LinearFMWaveform('SweepBandwidth',1e5, ...
    'PulseWidth',5e-5,'OutputFormat','Pulses', ...
    'FrequencyOffset',5e4,'CoefficientsOutputPort',true);
[wav,coeff] = waveform();
```

Create a matched filter that applies the coefficients as an input argument. Use the coefficients when applying the matched filter to the waveform. Plot the waveform and matched filter outputs.

```
mf = phased.MatchedFilter('CoefficientsSource','Input port');
mfOut = mf(wav,coeff);
subplot(211),plot(real(wav));
xlabel('Samples'),ylabel('Amplitude'),title('Waveform Output');
subplot(212),plot(abs(mfOut));
xlabel('Samples'),ylabel('Amplitude'),title('Matched Filter Output');
```



Compute Linear FM Bandwidth

Determine the bandwidth of a linear FM pulse waveform. The default value for an LFM waveform is 100 kHz.

```
waveform = phased.LinearFMWaveform;
bw = bandwidth(waveform)
```

```
bw = 100000
```

Version History

Introduced in R2011a

References

[1] Levanon, N. and E. Mozeson. *Radar Signals*. Hoboken, NJ: John Wiley & Sons, 2004.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `plot` method is not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also
`phased.NonlinearFMWaveform` | `phased.CustomFMWaveform` | `phased.PhaseCodedWaveform` | `phased.RectangularWaveform` | `phased.SteppedFMWaveform`

Topics
"Waveform Analysis Using the Ambiguity Function"

reset

System object: `phased.LinearFMWaveform`
Package: `phased`

Reset states of the linear FM waveform object

Syntax

`reset(H)`

Description

`reset(H)` resets the states of the `LinearFMWaveform` object, `H`. Afterward, if the `PRF` property is a vector, the next call to `step` uses the first PRF value in the vector.

step

System object: `phased.LinearFMWaveform`
Package: `phased`

Samples of linear FM pulse waveform

Syntax

```
Y = step(sLFM)
Y = step(sLFM,prfidx)
Y = step(sRFM,freqoffset)
[Y,PRF] = step( ___ )
[Y,COEFF] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

`Y = step(sLFM)` returns samples of the linear FM pulse in a column vector `Y`.

`Y = step(sLFM,prfidx)`, uses the `prfidx` index to select the PRF from the predefined vector of values specified by in the `PRF` property. This syntax applies when you set the `PRFSelectionInputPort` property to `true`.

`Y = step(sRFM,freqoffset)`, uses the `freqoffset` to generate the waveform with an offset as specified at step time. Use this syntax for cases where the transmit pulse frequency needs to be dynamically updated. This syntax applies when you set the `FrequencyOffsetSource` property to `'Input port'`.

`[Y,PRF] = step( ___ )` also returns the current pulse repetition frequency, `PRF`. To enable this syntax, set the `PRFOutputPort` property to `true` and set the `OutputFormat` property to `'Pulses'`.

`[Y,COEFF] = step( ___ )` returns the matched filter coefficients, `COEFF`, for the current pulse. To enable this syntax, set the `CoefficientsOutputPort` property to `true`. `COEFF` is returned as either an N_Z-by-1 vector or an N_Z-by-M matrix.

- An N_Z-by-1 vector is returned when:
  - The object has `OutputFormat` set to `'Pulses'` and `NumPulses` is equal to 1. N_Z is the pulse width.
  - The object is configured to generate constant pulse width waveforms (`DurationSpecification` is set to `'Pulse width'` or `'Duty cycle'` and `PRF` has one unique value); and either `OutputFormat` is set to `'Pulses'` and `NumPulses` is greater than 1, or the `OutputFormat` is set to `'Samples'`. For this case, N_Z is the pulse width.

- An N_Z-by-M matrix is returned when the object generates varying pulse widths (`DurationSpecification` property is set to `'Duty cycle'` and `PRF` has more than one unique value); and either `OutputFormat` set to `'Pulses'` and `NumPulses` is greater than 1, or `OutputFormat` is set to `'Samples'`. For this case, N_Z is the maximum of the pulse widths, and M is the number of unique PRFs.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example, `[Y,PRF,COEFF] = step(sRFM,prfidx,freqoffset)`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

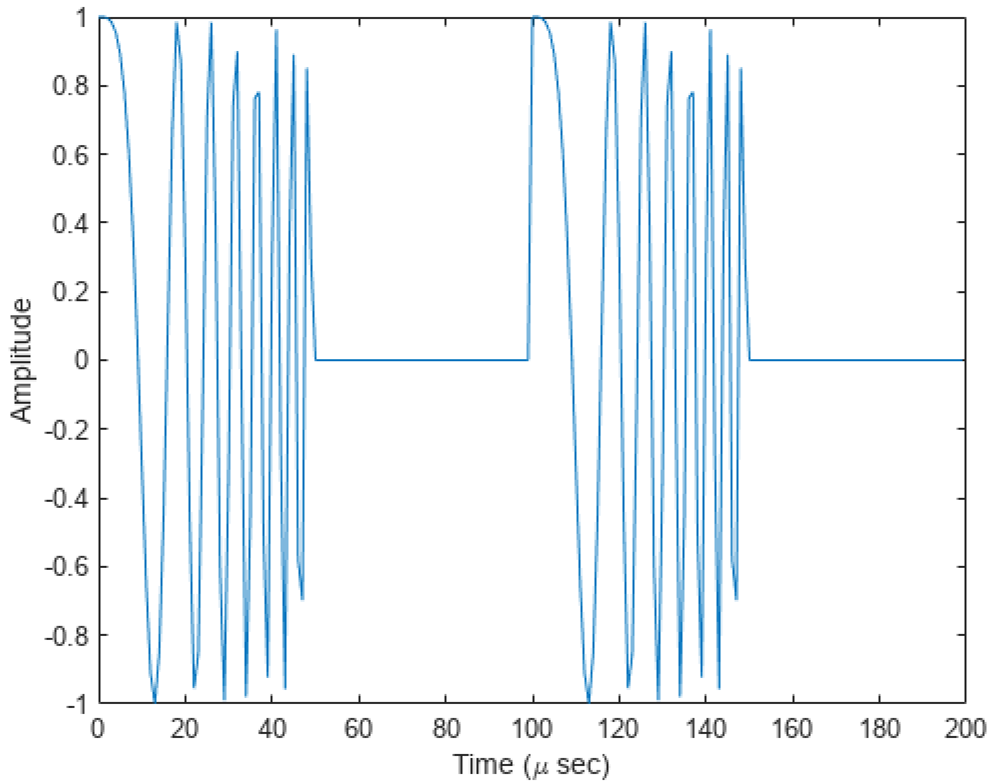
Create Linear FM Pulses

Construct a linear FM waveform having a sweep bandwidth of 300 kHz, a sample rate of 1 MHz, a pulse width of 50 microseconds, and a pulse repetition frequency of 10 kHz. Generate two pulses.

```
sLFM = phased.LinearFMWaveform('SweepBandwidth',3e5,...
    'OutputFormat','Pulses','SampleRate',1e6,...
    'PulseWidth',50e-6,'PRF',10e3,'NumPulses',2);
```

Obtain and plot the linear FM waveform.

```
wav = step(sLFM);
numpulses = size(wav,1);
t = [0:(numpulses-1)]/sLFM.SampleRate;
plot(t*1e6,real(wav))
xlabel('Time (\mu sec)')
ylabel('Amplitude')
```



Create Linear FM Pulses with Variable PRF

Construct six linear FM waveform pulses having a sweep bandwidth of 300 kHz, a sample rate of 1 MHz, a pulse width of 50 microseconds, and a duty cycle of 20%. Vary the pulse repetition frequency.

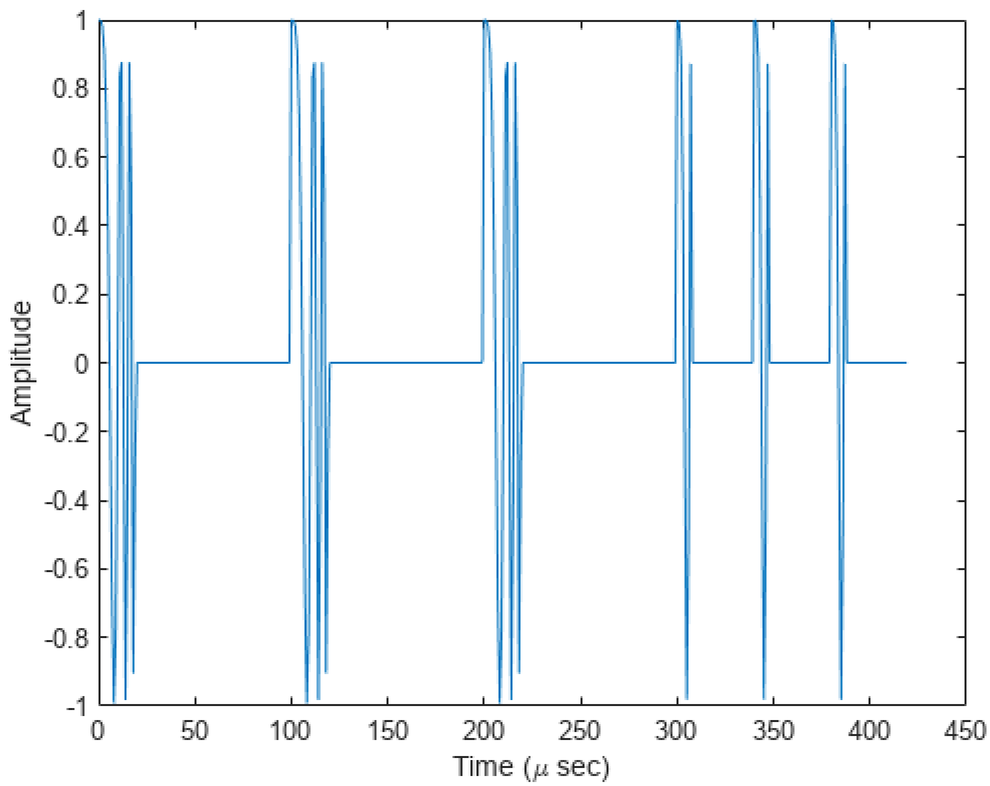
Set the sample rate and PRF. The ratio of sample rate to PRF must be an integer.

```
fs = 1e6;
PRF = [10000,25000];
sLFM = phased.LinearFMWaveform('SweepBandwidth',3e5,...
    'OutputFormat','Pulses','SampleRate',fs,...
    'DurationSpecification','Duty Cycle','DutyCycle',.2,...
    'PRF',PRF,'NumPulses',1,'PRFSelectionInputPort',true);
```

Obtain and plot the linear FM waveforms. For the first three calls to the step method, set the PRF to 10 kHz using the PRF index. For the next three calls, set the PRF to 25 kHz.

```
wav = [];
for n = 1:6
    idx = floor((n-1)/3)+1;
    wav1 = step(sLFM,idx);
    wav = [wav;wav1];
end
nsamps = size(wav,1);
t = [0:(nsamps-1)]/sLFM.SampleRate;
```

```
plot(t*1e6,real(wav))
xlabel('Time (\mu sec)')
ylabel('Amplitude')
```



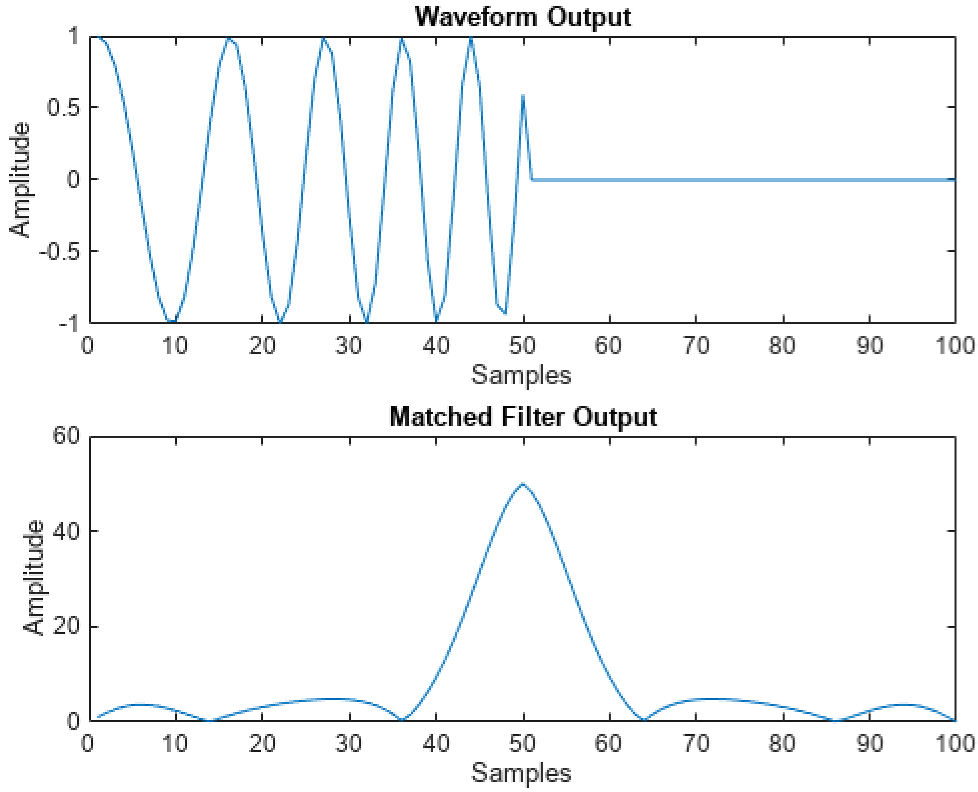
Generate Matched Filter Coefficients of Linear FM Pulse Waveform

Generate output samples and matched filter coefficients of a linear FM pulse waveform at a 50 kHz frequency offset.

```
waveform = phased.LinearFMWaveform('SweepBandwidth',1e5, ...
    'PulseWidth',5e-5,'OutputFormat','Pulses', ...
    'FrequencyOffset',5e4,'CoefficientsOutputPort',true);
[wav,coeff] = waveform();
```

Create a matched filter that applies the coefficients as an input argument. Use the coefficients when applying the matched filter to the waveform. Plot the waveform and matched filter outputs.

```
mf = phased.MatchedFilter('CoefficientsSource','Input port');
mfOut = mf(wav,coeff);
subplot(211),plot(real(wav));
xlabel('Samples'),ylabel('Amplitude'),title('Waveform Output');
subplot(212),plot(abs(mfOut));
xlabel('Samples'),ylabel('Amplitude'),title('Matched Filter Output');
```


Waveform Output
Amplitude
Samples
Matched Filter Output
Amplitude
Samples
1
0.5
0
-0.5
-1
60
40
20
0
0
10
20
30
40
50
60
70
80
90
100

phased.NonlinearFMWaveform

Package: `phased`

Nonlinear FM pulse waveform

Description

The `phased.NonlinearFMWaveform` System object creates a frequency-modulated waveform whose frequency is a nonlinear function of time (NLFM). NLFM waveforms achieve low-range sidelobes by shaping the spectrum using frequency modulation. Four different waveforms are supported depending on the `FrequencyModulation` property:

- `'Polynomial'` – Generate a waveform with an instantaneous frequency that follows a polynomial function.
- `'Hyperbolic'` – Generate a hyperbolic frequency modulated (HFM) waveform.
- `'Hybrid Linear-Tangent'` – Generate a hybrid NLFM waveform that combines an LFM waveform with tan-FM waveform.
- `'Stepped Price'` – Generate a stepped version of Price's NLFM waveform.

To create the waveforms:

1. Create the `phased.NonlinearFMWaveform` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
waveform = phased.NonlinearFMWaveform
waveform = phased.NonlinearFMWaveform(Name = Value)
```

Description

`waveform = phased.NonlinearFMWaveform` creates a nonlinear FM pulse `waveform` System object. By default, the waveform has a polynomial frequency modulation.

`waveform = phased.NonlinearFMWaveform(Name = Value)` creates a nonlinear FM pulse `waveform` System object with each specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1` = `Value1`,...,`NameN` = `ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

SampleRate — Signal sample rate
1e6 (default) | positive scalar

Signal sample rate, specified as a positive scalar. The ratio of sample rate to pulse repetition frequency must be a positive integer, so the number of samples in each pulse must be an integer value. Units are in Hertz.

Example: 100e3

Data Types: double

DurationSpecification — Method to set pulse duration
'Pulse width' (default) | 'Duty cycle'

Method to set pulse duration (pulse width), specified as 'Pulse width' or 'Duty cycle'. This property determines how you set the pulse duration.

- When you set this property to 'Pulse width', set the pulse duration directly using the PulseWidth property.
- When you set this property to 'Duty cycle', set the pulse duration from the values of the PRF and DutyCycle properties. The pulse width is equal to the value of the DutyCycle property divided by the value of the PRF property.

Data Types: char | string

PulseWidth — Pulse time duration
50e-6 (default) | positive scalar

Pulse time duration, specified as a positive scalar. The value must satisfy PulseWidth <= 1./PRF. Units are in seconds.

Dependencies

To enable this property, set the DurationSpecification property to 'Pulse width'.

Data Types: double

DutyCycle — Pulse duty cycle
0.5 (default) | positive scalar in the range [0,1]

Pulse duty cycle, specified as a positive scalar in the range [0,1]. The pulse width is the value of the DutyCycle property divided by the value of the PRF property. This quantity is dimensionless.

Example: 0.75

Dependencies

To enable this property, set the DurationSpecification property to 'Duty cycle'.

Data Types: double

PRF — Pulse repetition frequency
10e3 (default) | positive scalar | row vector of positive values

Pulse repetition frequency (PRF), specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval (PRI) is the inverse of the pulse repetition frequency PRF value. The PRF must satisfy these restrictions:

- The product of `PRF` and `PulseWidth` must be less than or equal to one. This condition requires that the pulse width is less than one PRI. For the phase-coded waveform, the pulse width is the product of the values of the `ChipWidth` and `NumChips` properties.
- The ratio of `SampleRate` to `PRF` must be an integer. This condition requires that the number of samples in one PRI is an integer.

You can set the value of PRF using the `PRF` property settings alone or using property settings in conjunction with the `prfidx` input argument of the object.

- When `PRFSelectionInputPort` is `false`, you set the PRF using the `PRF` properties alone. You can:
  - Implement a constant PRF by specifying the `PRF` property as a positive real-valued scalar.
  - Implement a staggered PRF by specifying the `PRF` property as a row vector with positive real-valued elements. Each call to the object uses successive elements of this vector as the PRF. Once the object reaches the last element of the vector, it continues the process cyclically with the first element of the vector.
- When `PRFSelectionInputPort` is `true`, you can set the PRF value using the `PRF` property in conjunction with the `prfidx` input argument. You implement a selectable PRF by specifying the `PRF` property as a row vector with positive real-valued elements. When you execute the object, the object selects a PRF by using the index you specify in the `prfidx` input argument to index into the PRF vector.

In all cases, the number of output samples is fixed when you set the `OutputFormat` property to `'Samples'`. When you use a varying PRF and also set the `OutputFormat` property to `'Pulses'`, the number of samples can vary.

Data Types: `double`

`PRFSelectionInputPort` — Enable PRF selection input
`false` (default) | `true`

Enable PRF selection input, specified as `false` or `true`. When you set this property to `true`, you can pass an index argument to the object to select a predefined value from the `PRF` property vector. When you set this property to `false`, the object uses the `PRF` property to define the PRF sequence used in the simulation.

Data Types: `logical`

`FrequencyModulation` — Frequency modulation
`'Polynomial'` (default) | `'Hyperbolic'` | `'Hybrid Linear-Tangent'` | `'Stepped Price'`

Frequency modulation of the nonlinear FM waveform, specified as `'Polynomial'` , `'Hyperbolic'`, `'Hybrid Linear-Tangent'`, or `'Stepped Price'`.

- When you set the property to `'Polynomial'` the System object generates a waveform with an instantaneous frequency that follows a polynomial function. You specify the coefficients in the `PolynomialCoefficients` property. The object normalizes the resulting frequency function such that each pulse sweeps the bandwidth you specify in `SweepBandwidth`. The `SweepDirection` property is inactive when `FrequencyModulation` is set to `'Polynomial'`'.

- When you set the property to 'Hyperbolic' the System object generates a hyperbolic frequency modulated (HFM) waveform. Use the HyperbolicStartFrequency property to set the start frequency of the hyperbolic sweep. The SweepInterval property is inactive in this case.
- When you set the property to 'Hybrid Linear-Tangent' System object generates a hybrid NLFM waveform that combines an LFM with a tan-FM as described by Collins and Atkins [1]. You specify the balance between LFM and tan-FM in the LinearTangentBalance property, and the portion of the tan(x) curve to used for tan-FM in the TangentCurvePortion property.
- When you set the property to 'Stepped Price' the System object generates a stepped version of the Price's NLFM waveform as given by Levanon and Mozeson [2]. The SweepBandwidth property is inactive in this case and the sweep bandwidth is determined by the bandwidth factors in the BandwidthFactors property and the number of frequency steps in the NumSteps property.

Example: 'Stepped Price'

Data Types: char | string

PolynomialCoefficients — Coefficients of polynomial frequency function

[1 0 0] (default) | (N+1)-length real-valued vector

Coefficients of the polynomial frequency function, specified as a length-(N+1) real-valued vector. The vector represents the coefficients of an N-th degree polynomial. The first entry in PolynomialCoefficients is the coefficient of the highest power N of the polynomial. The last entry is the coefficient of the power zero term of the polynomial.

Example: [0.5,1,1,0.5]

Dependencies

To enable this property, set the FrequencyModulation property to 'Polynomial'.

Data Types: double

LinearTangentBalance — Balance factor between linear FM and tan-FM functions

0.5 (default) | positive scalar

Balance factor between linear FM and tan-FM functions for a hybrid linear-tangent NLFM waveform, specified as a scalar. Units are dimensionless.

Example: 0.66

Dependencies

To enable this property, set the FrequencyModulation property to 'Hybrid Linear-Tangent'.

Data Types: double

TangentCurvePortion — Portion of tan(x) curve

1.4 (default) | real-valued scalar

Portion of the tan(x) curve between -π/2 and +π/2 to use as a tangent term in the hybrid linear-tangent FM frequency function, specified as a scalar between 0 and +π/2.

Example: 1.2

Dependencies

To enable this property, set the FrequencyModulation property to 'Hybrid Linear-Tangent'.

Data Types: `double`

`NumSteps` — Number of frequency steps
`50` (default) | positive integer

Number of frequency steps for the stepped form of Price's NLFM, specified as a positive integer.

Example: `10`

Dependencies

To enable this property, set the `FrequencyModulation` property to `'Stepped Price'`.

Data Types: `double`

`BandwidthFactors` — Bandwidth factors
`[1e4 2e4]` (default) | 2-element positive-valued vector

Bandwidth factors for the stepped form of Price's NLFM, specified as a two-element positive-valued vector of the form `[BL BC]`. `BL` determines the bandwidth of the linear component of the waveform and `BC` determines the bandwidth of the nonlinear component. The total sweep bandwidth is equal to $(M-1)/M*(BL+BC*M/\text{sqrt}(2*M-1))$, where M is the value of the `NumSteps` property. As M, increases the bandwidth of the linear component approximately equals BL, while the bandwidth of the nonlinear component grows with M.

Dependencies

To enable this property, set the `FrequencyModulation` property to `'Stepped Price'`.

Data Types: `double`

`HyperbolicStartFrequency` — Start frequency of hyperbolic FM waveform
`200e3` (default) | positive scalar

Start frequency of the hyperbolic frequency modulated waveform, specified as a positive scalar. Units are in Hz.

Dependencies

To enable this property, set the `FrequencyModulation` property to `'Hyperbolic'`.

Data Types: `double`

`SweepBandwidth` — Bandwidth of nonlinear FM sweep
`100e3` (default) | positive scalar

Bandwidth of nonlinear FM sweep, specified as a positive scalar. Units are in Hz.

Data Types: `double`

`SweepDirection` — Direction of nonlinear FM sweep
`'Up'` (default) | `'Down'`

Direction of the nonlinear FM sweep, specified as `'Up'` or `'Down'`.

Data Types: `char` | `string`

`SweepInterval` — Location of FM sweep interval
`'Positive'` (default) | `'Symmetric'`

Location of the FM sweep interval, specified as'Positive' or 'Symmetric'.

- If SweepInterval is 'Positive', the waveform sweeps the interval between 0 and *B* where *B* is the sweep bandwidth in the SweepBandwidth property.
- If SweepInterval is 'Symmetric', the waveform sweeps the interval between *-B/2* and *B/2*.

Data Types: char | string

Envelope — Waveform envelope function
'Rectangular' (default) | 'Gaussian' | 'Hamming' | 'Chebyshev' | 'Hann' | 'Kaiser' | 'Taylor' | 'Custom'

Waveform envelope function, specified as 'Rectangular', 'Gaussian', 'Hamming', 'Chebyshev', 'Hann', 'Kaiser', 'Taylor', or 'Custom'.

When Envelope is set to 'Custom' use the CustomEnvelope property to specify a custom envelope.

Example: 'Taylor'

Data Types: char | string

EnvelopeSidelobeLevel — Envelope window sidelobe level
30 (default) | positive scalar

Sidelobe level of a Kaiser, Chebyshev, or Taylor window used as the waveform envelope, specified as a positive scalar. Units are in dB.

Dependencies

To enable this property, set the Envelope property to 'Kaiser', 'Chebyshev', or 'Taylor'.

Data Types: double

CustomEnvelope — User-defined waveform envelope
@gausswin (default) | function handle | cell array

User-defined waveform envelope, specified as a function handle or cell array.

- If CustomEnvelope is a function handle, the specified function uses the window length as input and generates appropriate window coefficients.
- If CustomEnvelope is a cell array, then the first cell must be a function handle. The specified function takes the window length as the first input argument, with other additional input arguments if necessary, and generates appropriate window coefficients. The remaining entries in the cell array serve as additional input arguments to the function.

Example: {@chebwin,512,100}

Dependencies

To enable this property, set the Envelope property to 'Custom'.

Data Types: cell | function_handle

FrequencyOffsetSource — Source of frequency offset
'Property' (default) | 'Input port'

Source of frequency offset, specified as 'Property' or 'Input port'.

- When you set this property to `'Property'`, the frequency offset is determined by the value of the `FrequencyOffset` property.
- When you set this property to `'Input port'`, the frequency offset is determined by the input argument `freqoffset` when calling the object.

Example: `'Input port'`

Data Types: `char` | `string`

`FrequencyOffset` — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `150.0`

Dependencies

To enable this property, set the `FrequencyOffsetSource` property to `'Property'`.

Data Types: `double`

`OutputFormat` — Format of output signal
`'Pulses'` (default) | `'Samples'`

Format of output signal, specified as `'Pulses'` or `'Samples'`.

- When you set the `OutputFormat` property to `'Pulses'`, the output of the object takes the form of multiple pulses specified by the value of the `NumPulses` property. The number of samples per pulse can vary if you change the PRF during the simulation.
- When you set the `OutputFormat` property to `'Samples'`, the output of the object takes the form of multiple samples. In this case, the number of output signal samples is the value of the `NumSamples` property and is fixed.

Data Types: `char` | `string`

`NumSamples` — Number of samples in output
`100` (default) | positive integer

Number of samples in each output of the object, specified as a positive integer.

Dependencies

To enable this property, set the `OutputFormat` property to `'Samples'`.

Data Types: `double`

`NumPulses` — Number of pulses in output
`1` (default) | positive integer

Number of pulses in each output, specified as a positive integer.

Dependencies

To enable this property, set the `OutputFormat` property to `'Pulses'`.

Data Types: `double`

`PRFOutputPort` — Enable PRF output
`false` (default) | `true`

Enable PRF output, specified as `false` or `true`. Set this property to `true` to output the PRF.

Dependencies

To enable this property, set the `OutputFormat` property to `'Pulses'`.

Data Types: `logical`

`CoefficientsOutputPort` — Enable matched filter coefficients output
`false` (default) | `true`

Enable matched filter coefficients output, specified as `false` or `true`. Set this property to `true` to enable the object the output of the matched filter coefficients of the waveform used during the simulation.

Data Types: `logical`

Usage

Syntax

```
Y = waveform()
Y = waveform(prfidx)
Y = waveform(freqoffset)
[Y,PRF] = waveform( ___ )
[Y,coeff]= waveform()
```

Description

`Y = waveform()` returns samples of the nonlinear FM pulse in a column vector `Y`. `Y` can contain either a certain number of pulses or a certain number of samples.

`Y = waveform(prfidx)` specifies the index of the pulse repetition frequency (PRF), `prfidx`. The index identifies the entries specified in the `PRF` property. This syntax applies when you set the `PRFSelectionInputPort` property to `true`.

Use this syntax for the cases where the transmitted pulse needs to be dynamically selected. In such situations, the `PRF` property includes a list of predetermined choices of PRF's. During the simulation, using `prfidx`, one of the PRFs is selected as the PRF for the next transmission.

Note that the transmission always finishes the current pulse before starting the next pulse. Therefore, when you set the `OutputFormat` property to `'Samples'` and then specify the `NumSamples` property to be shorter than a pulse, it is possible that during a given simulation step, if the entire output is needed to finish the previously transmitted pulse, the specified `prfidx` is ignored.

`Y = waveform(freqoffset)` specifies the value of the frequency offset `freqoffset` as a finite real value. The offset is used to generate the waveform with a frequency offset. Use this syntax for the cases where the transmit pulse frequency needs to be dynamically updated. To enable this syntax set the `FrequencyOffsetSource` property to `'Input port'`.

`[Y,PRF] = waveform( ___ )` also returns the current pulse repetition frequency, `PRF`. To enable this syntax, set the `PRFOutputPort` property to `true` and set the `OutputFormat` property to `'Pulses'`.

`[Y,coeff]= waveform()` returns an additional output `coeff`, as the matched filter coefficients. To use this syntax, set the `CoefficientsOutputPort` property to `true`.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example,

```
[Y,PRF,coeff] = waveform(prfidx,freqoffset)
```

Input Arguments

`prfidx` — index of pulse repetition frequency
positive integer

Index of pulse repetition frequency, specified as a positive integer. The index identifies the entries in the `PRF` property. This syntax applies when you set the `PRFSelectionInputPort` property to `true`. Use this argument for the case when the transmit pulse needs to be dynamically selected. Under such situations, the PRF property includes a list of predetermined choices of PRFs. During the simulation, based on the PRF index input, one of the PRFs is selected as the PRF for the next transmission.

`freqoffset` — frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. This argument generates the waveform with a frequency offset when the object is called. Use this syntax for the cases where the transmit pulse frequency needs to be dynamically updated.

Dependencies

To use this argument, set the `FrequencyOffsetSource` property to `'Input port'`.

Data Types: `double`

Output Arguments

PRF — pulse repetition frequency
scalar

Pulse repetition frequency, returned as a scalar. Returns the current PRF used by the object. Units are in Hz.

Dependencies

To enable this argument, set the `PRFOutputPort` property to `true` and set the `OutputFormat` to `'Pulses'`.

`coeff` — matched filter coefficients
N_Z-by-1 complex-valued vector | N_Z-by-M complex-valued matrix

Matched filter coefficients, returned as an N_Z-by-1 complex-valued vector or an N_Z-by-M complex-valued matrix.

- If you set `OutputFormat` to `'Pulses'` and `NumPulses` is 1, the object returns `coeff` as an N_Z-length vector. N_Z corresponds to the pulse width.
- If you set `OutputFormat` to `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is `'Samples'` and `DurationSpecification` is `'Pulse width'`, `coeff` is returned as an N_Z-length vector. N_Z corresponds to the pulse width.
- If `OutputFormat` is set to `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is set to `'Samples'` and `DurationSpecification` is set to `'Duty cycle'` with only one unique PRF value, `coeff` is returned as an N_Z-length vector. N_Z corresponds to the pulse width.
- If `OutputFormat` is `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is `'Samples'` and `DurationSpecification` is `'Duty cycle'`, `coeff` is returned as an N_Z-by-M matrix. N_Z corresponds to the maximum pulse width and M corresponds to the number of unique PRFs.

Dependencies

To enable this argument, set the `CoefficientsOutputPort` property to `true`.

Data Types: `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to waveform System objects

bandwidth	Waveform bandwidth
getMatchedFilter	Matched filter coefficients derived from waveform
plot	Plot pulse waveform

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Create Nonlinear FM Waveform with Default Properties

Create a nonlinear FM waveform with the default polynomial frequency modulation.

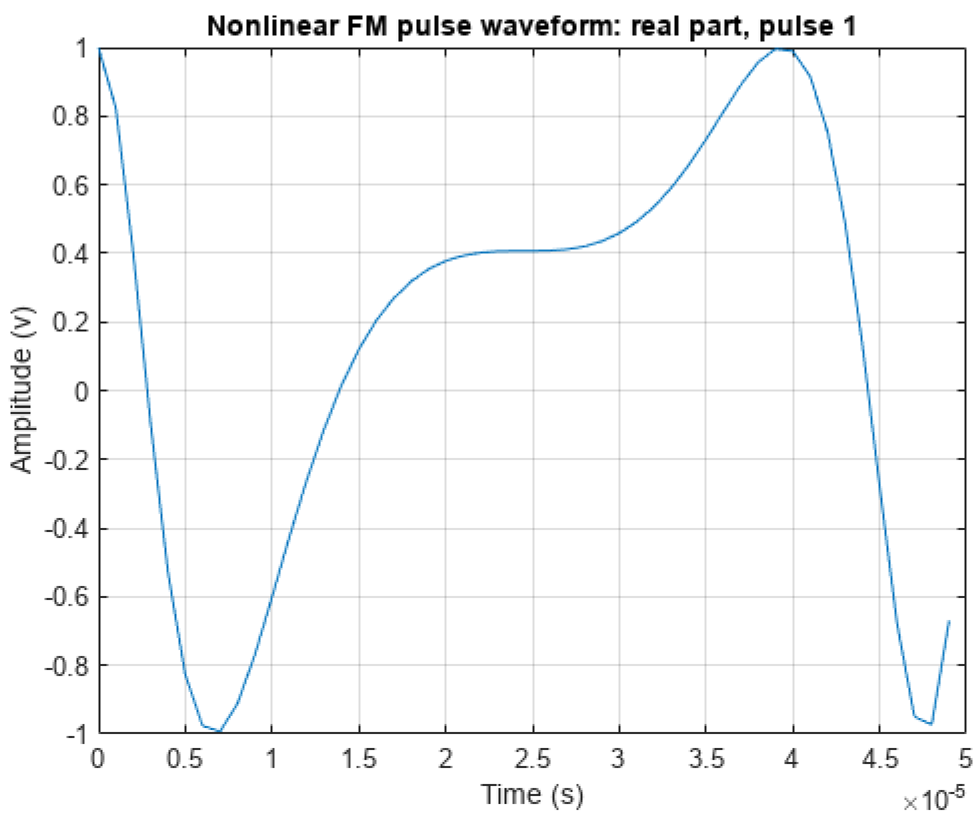
```
waveform = phased.NonlinearFMWaveform()
```

```
waveform = 
  phased.NonlinearFMWaveform with properties:

               SampleRate: 1000000
    DurationSpecification: 'Pulse width'
               PulseWidth: 5.0000e-05
```

```
                    PRF: 10000
  PRFSelectionInputPort: false
    FrequencyModulation: 'Polynomial'
 PolynomialCoefficients: [1 0 0]
         SweepBandwidth: 100000
          SweepInterval: 'Positive'
               Envelope: 'Rectangular'
  FrequencyOffsetSource: 'Property'
        FrequencyOffset: 0
           OutputFormat: 'Pulses'
              NumPulses: 1
          PRFOutputPort: false
 CoefficientsOutputPort: false
```

Display the real part of the waveform.

```
plot(waveform)
```

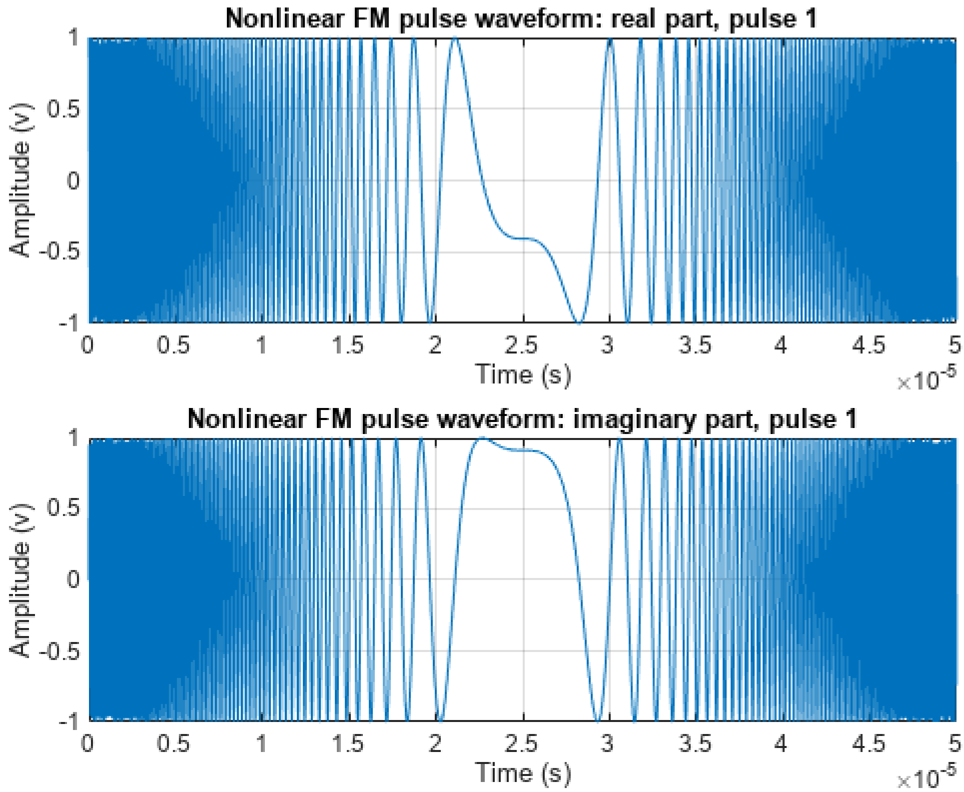


Plot Quadratic FM Pulse Waveform

Create and plot a quadratic FM pulse waveform. The pulse has a 10 MHz bandwidth and 50 μsec duration. The pulse sample rate is 10 times the bandwidth.

```
BW = 10e6;
T = 50e-6;
```

```
waveform = phased.NonlinearFMWaveform( ...
    'SampleRate',10*BW,'SweepBandwidth',BW, ...
    'PulseWidth',T);
plot(waveform,PlotType='complex')
```



Create Hybrid Linear-Tangent FM Waveform

Generate samples of a hybrid linear-tangent FM waveform with a bandwidth of 2 kHz and a pulse width of 100 ms. Set the pulse repetition frequency to 5 Hz and the sample frequency to 20 kHz. Set the factor controlling the balance between the linear and the nonlinear terms to 0.5. Set the factor controlling the portion of the tan(x) curve used for frequency modulation to 1.4. Plot the autocorrelation function to determine the resulting range sidelobe level.

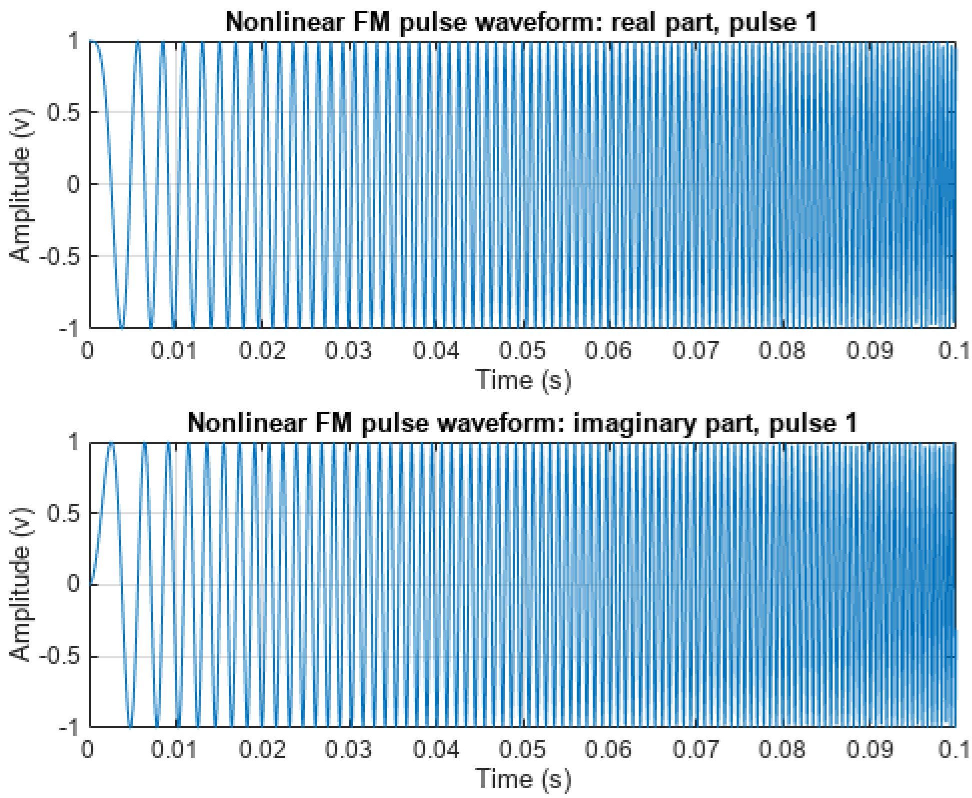
```
BW = 2000;
T = 100e-3;
PRF = 5;
fs = 10*BW;
```

Set the balance between tan-FM and LFM to 0.5 and set the portion of tan(x) to use to 1.4.

```
alpha = 0.5;
gamma = 1.4;
```

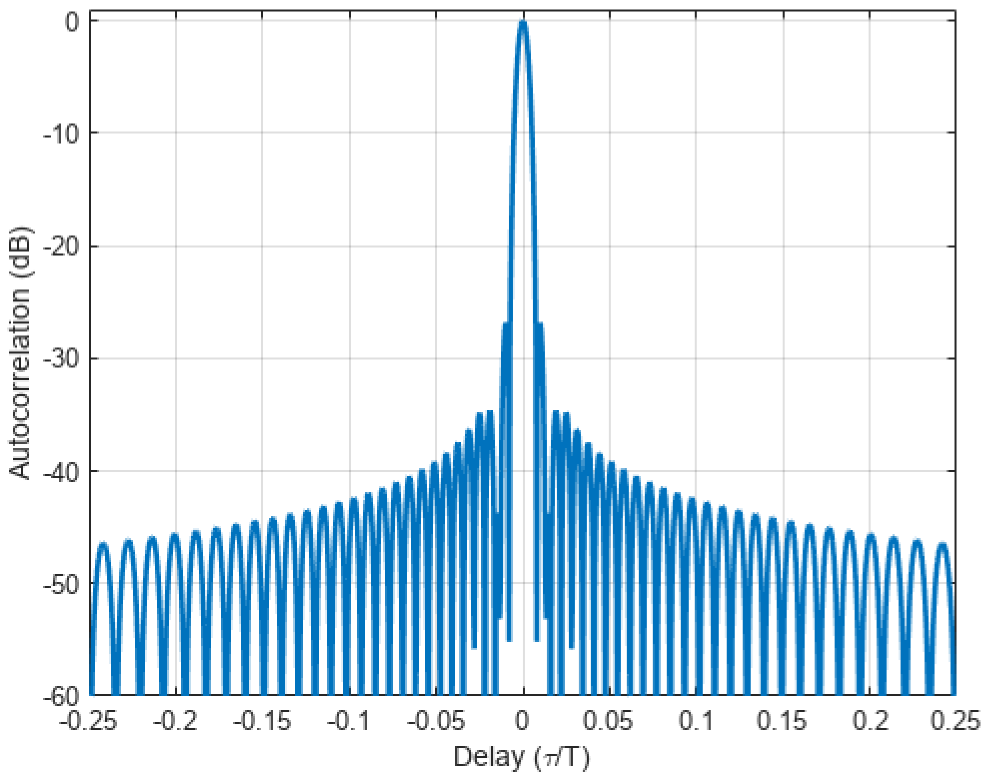
Create the waveform and plot the real and imaginary parts.

```
waveform = phased.NonlinearFMWaveform('SampleRate',fs,'PulseWidth',T, ...
    'PRF',PRF,'FrequencyModulation','Hybrid Linear-Tangent', ...
    'LinearTangentBalance',alpha,'TangentCurvePortion',gamma, ...
    'SweepBandwidth',BW,'OutputFormat','Pulses','NumPulses',2);
wav = waveform();
figure
plot(waveform,PlotType='complex')
```



Compute and plot the autocorrelation function.

```
[acf,delay] = ambgfun(wav,waveform.SampleRate,waveform.PRF, ...
    Cut='Doppler');
figure
plot(delay/waveform.PulseWidth,mag2db(acf),LineWidth=2)
grid on
xlim([-0.25 0.25])
ylim([-60 1])
xlabel('Delay (\tau/T)')
ylabel('Autocorrelation (dB)')
```



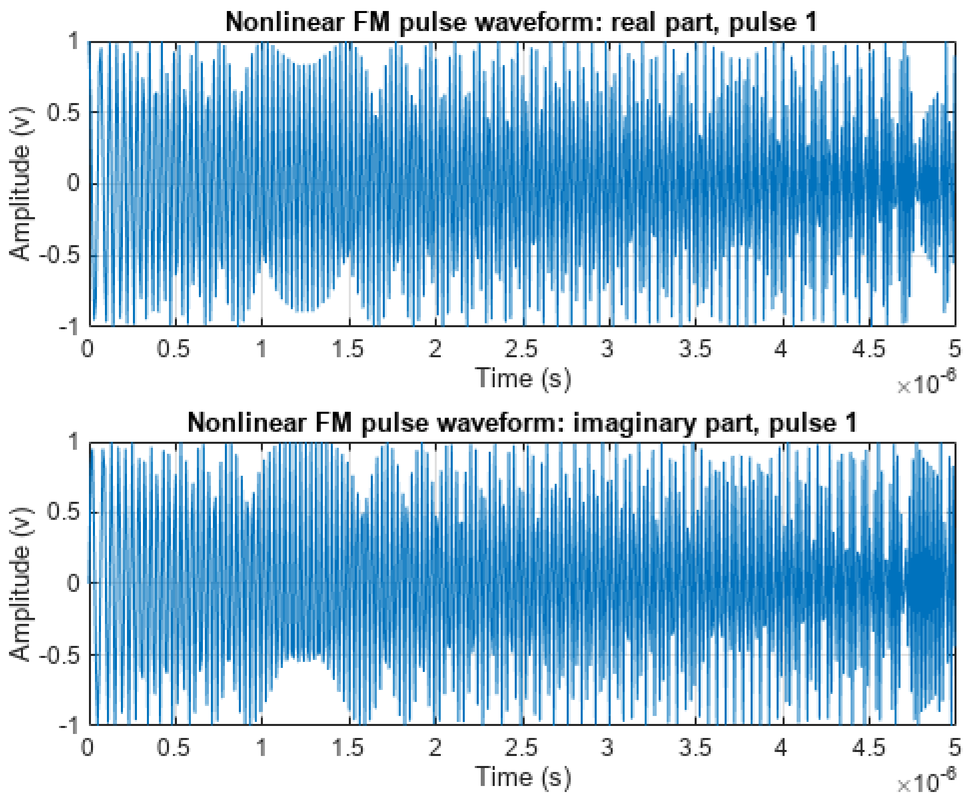
Plot Stepped Version of Price's Waveform

Generate output samples of a nonlinear FM pulse waveform based on the stepped version of Price's waveform. The pulse width is 5e-6 seconds. The waveform performs 50 frequency steps within one pulse. The time-bandwidth products of the linear and nonlinear components are 20 and 40 respectively. The start frequency of the sweep is 15 MHz. Generate matched filter coefficients and then apply the matched filter.

```
T = 5e-6;
M = 50;
BLT = 20;
BCT = 40;
bfs = [BLT BCT]/T;
fs = 100e6;
fstart = 15e6;

waveform = phased.NonlinearFMWaveform(SampleRate=fs, ...
    PulseWidth=T,FrequencyModulation='Stepped Price', ...
    BandwidthFactors=bfs,NumSteps=M,FrequencyOffset=fstart, ...
    OutputFormat='Pulses',CoefficientsOutputPort=true);
```

Plot the complex waveform.

```
figure
plot(waveform,PlotType='complex')
```



Obtain the resulting sweep bandwidth (Hz)

```
bandwidth(waveform)
```

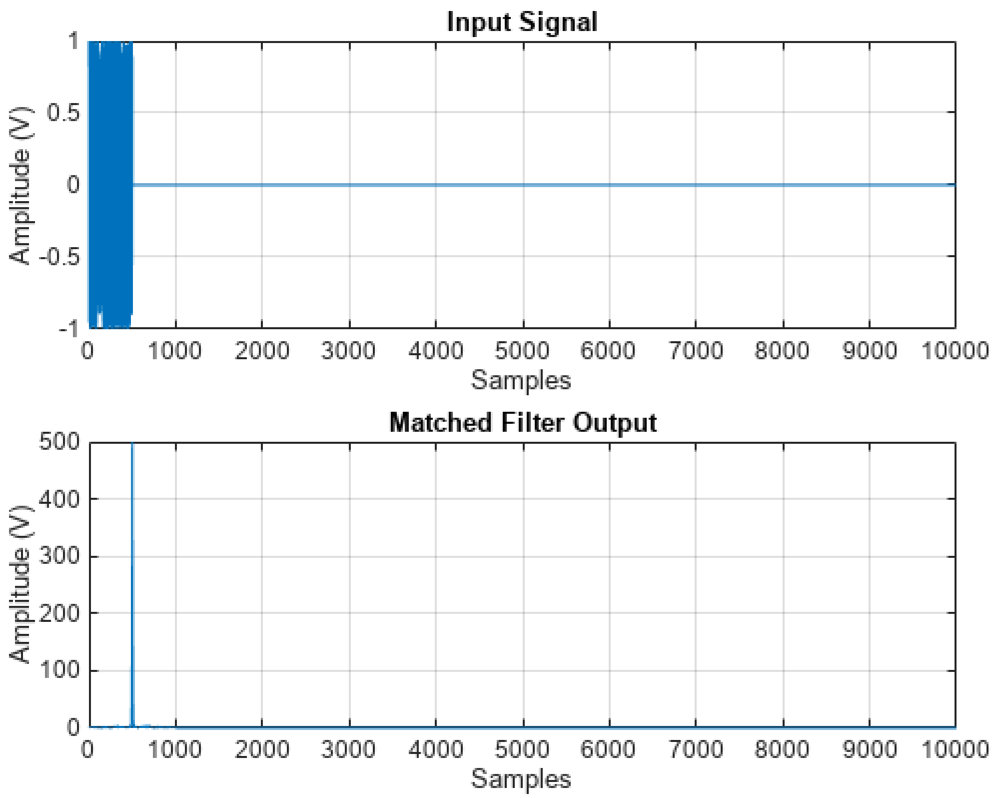
```
ans = 4.3317e+07
```

Find the matched filter coefficients. Then plot the filtered waveform.

```
[wav,coeff] = waveform();
mf = phased.MatchedFilter(CoefficientsSource='Input port');
mfout = mf(wav,coeff);
```

Plot the filter input and output.

```
figure
subplot(211)
plot(real(wav))
grid on
xlabel('Samples')
ylabel('Amplitude (V)')
title('Input Signal')

subplot(212)
plot(abs(mfout));
grid on
xlabel('Samples')
ylabel('Amplitude (V)')
title('Matched Filter Output')
```



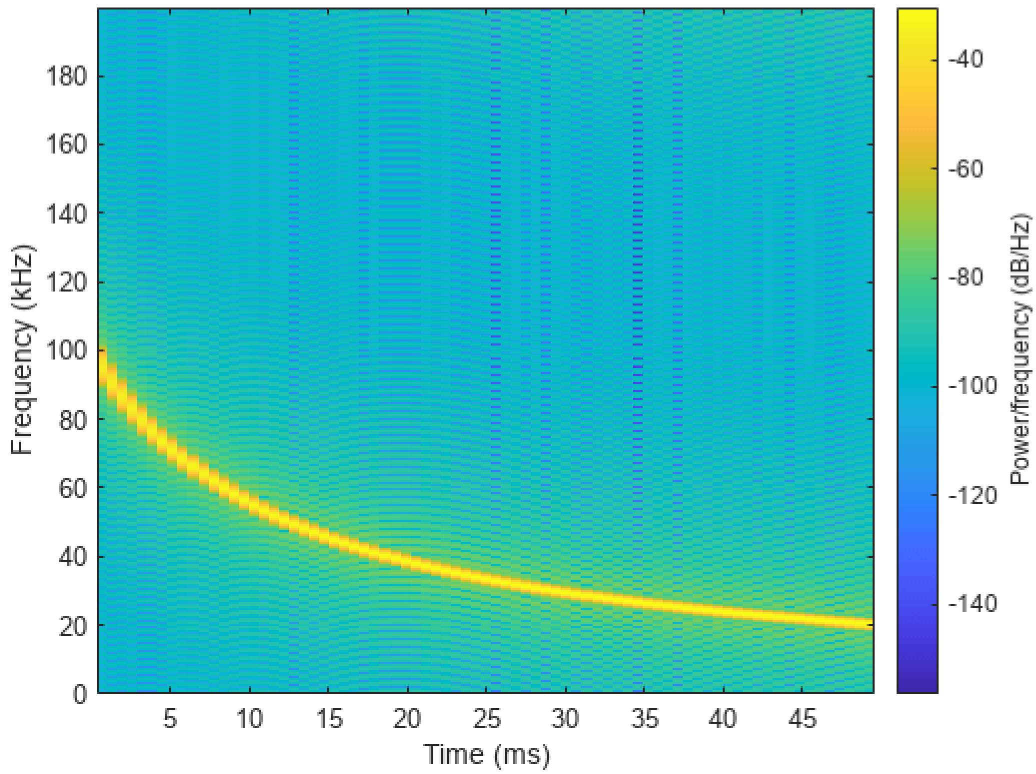
Plot Spectrogram of Hyperbolic FM Waveform

Create a hyperbolic frequency modulated (HFM) waveform. The start and the stop frequencies of the sweep are 100 kHz and 20 kHz, respectively. The pulse width is 50 ms. The sample rate is 200 kHz. Plot the spectrogram.

```
f1 = 100e3;
f2 = 20e3;
BW = f1 - f2;
T = 50e-3;
fs = 2*f1;
```

Create a hyperbolic FM waveform.

```
waveform = phased.NonlinearFMWaveform( ...
    SampleRate=fs,PulseWidth=T,SweepBandwidth=BW, ...
    FrequencyModulation='Hyperbolic', ...
    HyperbolicStartFrequency=f1, ...
    SweepDirection='down',PRF=1/T);
sig = waveform();
spectrogram(sig, 256, 128, 1024, fs, 'yaxis')
```



Version History

Introduced in R2023a

References

[1] Collins, T., and P. Atkins. "Nonlinear frequency modulation chirps for active sonar." *IEE Proceedings-Radar, Sonar and Navigation* 146.6 (1999): 312-316.

[2] Levanon, Nadav, and Eli Mozeson. *Radar signals*. John Wiley & Sons, 2004, pp. 92-93.

[3] Doerry, Armin Walter. "Generating nonlinear FM chirp waveforms for radar". No. SAND2006-5856. *Sandia National Laboratories (SNL)*, Albuquerque, NM, and Livermore, CA (United States), 2006.

[4] Cook, C. E. "A class of nonlinear FM pulse compression signals." *Proceedings of the IEEE* 52.11 (1964): 1369-1371.

[5] Yang, J., and T. K. Sarkar. "Doppler-invariant property of hyperbolic frequency modulated waveforms." *Microwave and optical technology letters* 48.6 (2006): 1174-1179.

[6] Melvin, William L., and James Scheer. *Principles of modern radar: advanced techniques.* SciTech Pub., 2013.

[7] Alphonse, Sebastian, and Geoffrey A. Williamson. "Evaluation of a class of NLFM radar signals." *EURASIP Journal on Advances in Signal Processing* 2019.1 (2019): 1-12.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.LinearFMWaveform` | `phased.CustomFMWaveform` | `phased.PhaseCodedWaveform` | `phased.RectangularWaveform` | `phased.SteppedFMWaveform` | `nlfmspec2freq`

Topics

“Waveform Analysis Using the Ambiguity Function”

phased.CustomFMWaveform

Package: `phased`

Custom FM pulse waveform

Description

The `phased.CustomFMWaveform` System object lets you define a waveform with a user-defined frequency modulation (FM) and waveform envelope.

To create the waveform:

1. Create the `phased.CustomFMWaveform` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
waveform = phased.CustomFMWaveform()
waveform = phased.CustomFMWaveform(Name=Value)
```

Description

`waveform = phased.CustomFMWaveform()` creates a custom FM pulse `waveform` System object with linear frequency modulation and a rectangular envelope.

`waveform = phased.CustomFMWaveform(Name=Value)` sets additional properties using `Name-Value` arguments. You can specify additional name-value pair arguments in any order as (`Neme1=Value1`,...,`NameN=ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SampleRate` — Signal sample rate
`1e6` (default) | positive scalar

Signal sample rate, specified as a positive scalar. The ratio of sample rate to pulse repetition frequency must be a positive integer, so the number of samples in each pulse must be an integer value. Units are in Hertz.

Example: 100e3

Data Types: `double`

`DurationSpecification` — Method to set pulse duration
`'Pulse width'` (default) | `'Duty cycle'`

Method to set pulse duration (pulse width), specified as `'Pulse width'` or `'Duty cycle'`. This property determines how you set the pulse duration.

- When you set this property to `'Pulse width'`, set the pulse duration directly using the `PulseWidth` property.
- When you set this property to `'Duty cycle'`, set the pulse duration from the values of the `PRF` and `DutyCycle` properties. The pulse width is equal to the value of the `DutyCycle` property divided by the value of the `PRF` property.

Data Types: `char` | `string`

`PulseWidth` — Pulse time duration
`50e-6` (default) | positive scalar

Pulse time duration, specified as a positive scalar. The value must satisfy `PulseWidth <= 1./PRF`. Units are in seconds.

Dependencies

To enable this property, set the `DurationSpecification` property to `'Pulse width'`.

Data Types: `double`

`DutyCycle` — Pulse duty cycle
`0.5` (default) | positive scalar in the range `[0,1]`

Pulse duty cycle, specified as a positive scalar in the range `[0,1]`. The pulse width is the value of the `DutyCycle` property divided by the value of the `PRF` property. This quantity is dimensionless.

Example: `0.75`

Dependencies

To enable this property, set the `DurationSpecification` property to `'Duty cycle'`.

Data Types: `double`

`PRF` — Pulse repetition frequency
`10e3` (default) | positive scalar | row vector of positive values

Pulse repetition frequency (PRF), specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval (PRI) is the inverse of the pulse repetition frequency PRF value. The PRF must satisfy these restrictions:

- The product of `PRF` and `PulseWidth` must be less than or equal to one. This condition requires that the pulse width is less than one PRI. For the phase-coded waveform, the pulse width is the product of the values of the `ChipWidth` and `NumChips` properties.
- The ratio of `SampleRate` to `PRF` must be an integer. This condition requires that the number of samples in one PRI is an integer.

You can set the value of PRF using the `PRF` property settings alone or using property settings in conjunction with the `prfidx` input argument of the object.

- When `PRFSelectionInputPort` is `false`, you set the PRF using the `PRF` properties alone. You can:
  - Implement a constant PRF by specifying the `PRF` property as a positive real-valued scalar.
  - Implement a staggered PRF by specifying the `PRF` property as a row vector with positive real-valued elements. Each call to the object uses successive elements of this vector as the PRF. Once the object reaches the last element of the vector, it continues the process cyclically with the first element of the vector.
- When `PRFSelectionInputPort` is `true`, you can set the PRF value using the `PRF` property in conjunction with the `prfidx` input argument. You implement a selectable PRF by specifying the `PRF` property as a row vector with positive real-valued elements. When you execute the object, the object selects a PRF by using the index you specify in the `prfidx` input argument to index into the PRF vector.

In all cases, the number of output samples is fixed when you set the `OutputFormat` property to `'Samples'`. When you use a varying PRF and also set the `OutputFormat` property to `'Pulses'`, the number of samples can vary.

Data Types: `double`

`PRFSelectionInputPort` — Enable PRF selection input
`false` (default) | `true`

Enable PRF selection input, specified as `false` or `true`. When you set this property to `true`, you can pass an index argument to the object to select a predefined value from the `PRF` property vector. When you set this property to `false`, the object uses the `PRF` property to define the PRF sequence used in the simulation.

Data Types: `logical`

`FrequencyModulation` — Waveform frequency modulation
length-*M* real-valued vector | function handle

Waveform frequency modulation, specified as a length-*M* real-valued vector, function handle, or cell array.

- If the `FrequencyModulation` property is a vector, it specifies samples of the instantaneous frequency at M points as $[f_1, f_2, \ldots, f_M]$. The waveform sweeps the specified frequencies such that for the k^{th} pulse with start time t_k and duration T_k, the instantaneous frequency at time

  $$t_m = t_k + (m-1)T_k/(M-1)$$

  is equal to f_m where $t_k \leqq t_m \leqq t_k + T_k$ and $m = 1 \ldots M$. The instantaneous frequencies between time t_m and t_{m+1} are found by linearly interpolating between f_m and f_{m+1}. The resulting custom FM waveform is a piecewise linear FM (LFM) waveform consisting of M-1 LFM sections of equal duration.
- If the `FrequencyModulation` property is a function handle, the function must have the following syntax: `f = fmFcn(t)` where `f` is the instantaneous frequency at time `t`. `t` is the time at which to compute the instantaneous frequency. The values in `t` are between 0 and the pulse width.

- If the `FrequencyModulation` property is a cell array, then the first cell must be a function handle as specified above. The remaining entries in the cell array are the additional input arguments to the function, if any.

Data Types: `double`

`Envelope` — Waveform envelope function
`'Rectangular'` (default) | `'Gaussian'` | `'Hamming'` | `'Chebyshev'` | `'Hann'` | `'Kaiser'` | `'Taylor'` | `'Custom'`

Waveform envelope function, specified as `'Rectangular'`, `'Gaussian'`, `'Hamming'`, `'Chebyshev'`, `'Hann'`, `'Kaiser'`, `'Taylor'`, or `'Custom'`.

When you set `Envelope` is set to `'Custom'` must use the `CustomEnvelope` property to specify a custom envelope.

Example: `'Taylor'`

Data Types: `char` | `string`

`EnvelopeSidelobeLevel` — Envelope window sidelobe level
`30` (default) | positive scalar

Sidelobe level for a Kaiser, Chebyshev, or Taylor window used as the waveform envelope, specified as a positive scalar. Units are in dB.

Dependencies

To enable this property, set the `Envelope` property to `'Kaiser'`, `'Chebyshev'`, or `'Taylor'`.

Data Types: `double`

`CustomEnvelope` — User-defined waveform envelope
`@gausswin` (default) | function handle | cell array

User-defined waveform envelope, specified as a function handle or cell array.

- If `CustomEnvelope` is a function handle, the specified function uses the window length as input and generates appropriate window coefficients.
- If `CustomEnvelope` is a cell array, then the first cell must be a function handle. The specified function takes the window length as the first input argument, with other additional input arguments if necessary, and generates appropriate window coefficients. The remaining entries in the cell array serve as additional input arguments to the function.

Example: `{@chebwin,512,100}`

Dependencies

To enable this property, set the `Envelope` property to `'Custom'`.

Data Types: `cell` | `function_handle`

`FrequencyOffsetSource` — Source of frequency offset
`'Property'` (default) | `'Input port'`

Source of frequency offset, specified as `'Property'` or `'Input port'`.

- When you set this property to `'Property'`, the frequency offset is determined by the value of the `FrequencyOffset` property.
- When you set this property to `'Input port'`, the frequency offset is determined by the input argument `freqoffset` when calling the object.

Example: `'Input port'`

Data Types: `char` | `string`

`FrequencyOffset` — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `150.0`

Dependencies

To enable this property, set the `FrequencyOffsetSource` property to `'Property'`.

Data Types: `double`

`OutputFormat` — Format of output signal
`'Pulses'` (default) | `'Samples'`

Format of output signal, specified as `'Pulses'` or `'Samples'`.

- When you set the `OutputFormat` property to `'Pulses'`, the output of the object takes the form of multiple pulses specified by the value of the `NumPulses` property. The number of samples per pulse can vary if you change the PRF during the simulation.
- When you set the `OutputFormat` property to `'Samples'`, the output of the object takes the form of multiple samples. In this case, the number of output signal samples is the value of the `NumSamples` property and is fixed.

Data Types: `char` | `string`

`NumSamples` — Number of samples in output
`100` (default) | positive integer

Number of samples in each output of the object, specified as a positive integer.

Dependencies

To enable this property, set the `OutputFormat` property to `'Samples'`.

Data Types: `double`

`NumPulses` — Number of pulses in output
`1` (default) | positive integer

Number of pulses in each output, specified as a positive integer.

Dependencies

To enable this property, set the `OutputFormat` property to `'Pulses'`.

Data Types: `double`

`PRFOutputPort` — Enable PRF output
`false` (default) | `true`

Enable PRF output, specified as `false` or `true`. Set this property to `true` to output the PRF.

Dependencies

To enable this property, set the `OutputFormat` property to `'Pulses'`.

Data Types: `logical`

`CoefficientsOutputPort` — Enable matched filter coefficients output
`false` (default) | `true`

Enable matched filter coefficients output, specified as `false` or `true`. Set this property to `true` to enable the object the output of the matched filter coefficients of the waveform used during the simulation.

Data Types: `logical`

Usage

Syntax

```
Y = waveform()
Y = waveform(prfidx)
Y = waveform(freqoffset)
[Y,PRF] = waveform(___)
[Y,coeff]= waveform()
```

Description

`Y = waveform()` returns samples of the custom nonlinear FM pulse in a column vector `Y`. `Y` can contain a certain number of pulses or a certain number of samples.

`Y = waveform(prfidx)` specifies the index of the PRF vector, `prfidx`. The object uses the index is used to identify the entries specified in the PRF property. To enable this syntax, set the `PRFSelectionInputPort` property to `true`.

Use this syntax for the cases where you need to dynamically select the transmitted pulse. In such situations, the `PRF` property includes a list of predetermined PRF values. During the simulation, based on PRF index input, the object selects one of the PRFs values for the PRF for the next transmission.

The transmission always finishes the current pulse before starting the next pulse. Therefore, when you set the `OutputFormat` property to `'Samples'` and then specify the `NumSamples` property to be shorter than a pulse, the object can ignore the PRF index during a given simulation step if needs the entire output to finish the previously transmitted pulse.

`Y = waveform(freqoffset)` specifies the value of the frequency offset `freqoffset`. The offset generates the waveform with a frequency offset . Use this syntax when you need to update the transmit pulse frequency dynamically. To enable this syntax, set the `FrequencyOffsetSource` property to `'Input port'`.

`[Y,PRF] = waveform( ___ )` also returns the current `PRF`. To enable this syntax, set the `PRFOutputPort` property to `true` and set the `OutputFormat` property to `'Pulses'`.

`[Y,coeff]= waveform()` returns the matched filter coefficients `coeff` when you set the `CoefficientsOutputPort` property to `true`.

You can combine optional input and output arguments when you set their enabling properties are set. List optional inputs and outputs in the same order as the order of the enabling properties. For example,

```
[Y,PRF,coeff] = waveform(prfidx,freqoffset)
```

Input Arguments

`prfidx` — index of PRF

positive integer

Index of PRF, specified as a positive integer. The index identifies the entries in the `PRF` property. To enable this syntax, set the `PRFSelectionInputPort` property to `true`. Use this syntax when you need to dynamically select the transmit pulse. In such situations, the PRF property includes a list of predetermined values. During the simulation, based on the PRF index input, the object selects one of the PRFs is selected as the PRF for the next transmission.

`freqoffset` — frequency offset

`0` (default) | scalar

Frequency offset, specified as a finite scalar. The object generates the waveform with a frequency offset. Use this syntax when you need to update the transmit pulse frequency dynamically.

Dependencies

To use this argument, set the `FrequencyOffsetSource` property to `'Input port'`.

Data Types: `double`

Output Arguments

PRF — pulse repetition frequency

scalar

Pulse repetition frequency, returned as a scalar. When you set the `PRFOutputPort` property to true it returns the current PRF used by the system. Units are in Hz.

Dependencies

To enable this argument, set the `PRFOutputPort` property to `true` and set the `OutputFormat` to `'Pulses'`.

`coeff` — matched filter coefficients

N_Z-by-1 complex-valued vector | N_Z-by-M complex-valued matrix

Matched filter coefficients, returned as an N_Z-by-1 complex-valued vector or an N_Z-by-M complex-valued matrix.

- If you set `OutputFormat` to `'Pulses'` and `NumPulses` is 1, the object returns `coeff` as an N_Z-length vector. N_Z corresponds to the pulse width.

- If you set `OutputFormat` to `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is `'Samples'` and `DurationSpecification` is `'Pulse width'`, `coeff` is returned as an N_Z-length vector. N_Z corresponds to the pulse width.
- If `OutputFormat` is set to `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is set to `'Samples'` and `DurationSpecification` is set to `'Duty cycle'` with only one unique PRF value, `coeff` is returned as an N_Z-length vector. N_Z corresponds to the pulse width.
- If `OutputFormat` is `'Pulses'` with `NumPulses` greater than 1 or `OutputFormat` is `'Samples'` and `DurationSpecification` is `'Duty cycle'`, `coeff` is returned as an N_Z-by-M matrix. N_Z corresponds to the maximum pulse width and M corresponds to the number of unique PRFs.

Dependencies

To enable this argument, set the `CoefficientsOutputPort` property to `true`.

Data Types: `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to waveform System objects

bandwidth	Waveform bandwidth
getMatchedFilter	Matched filter coefficients derived from waveform
plot	Plot pulse waveform

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Create Custom FM Waveform with Default Properties

Create a custom FM waveform with the default frequency modulation and envelope function.

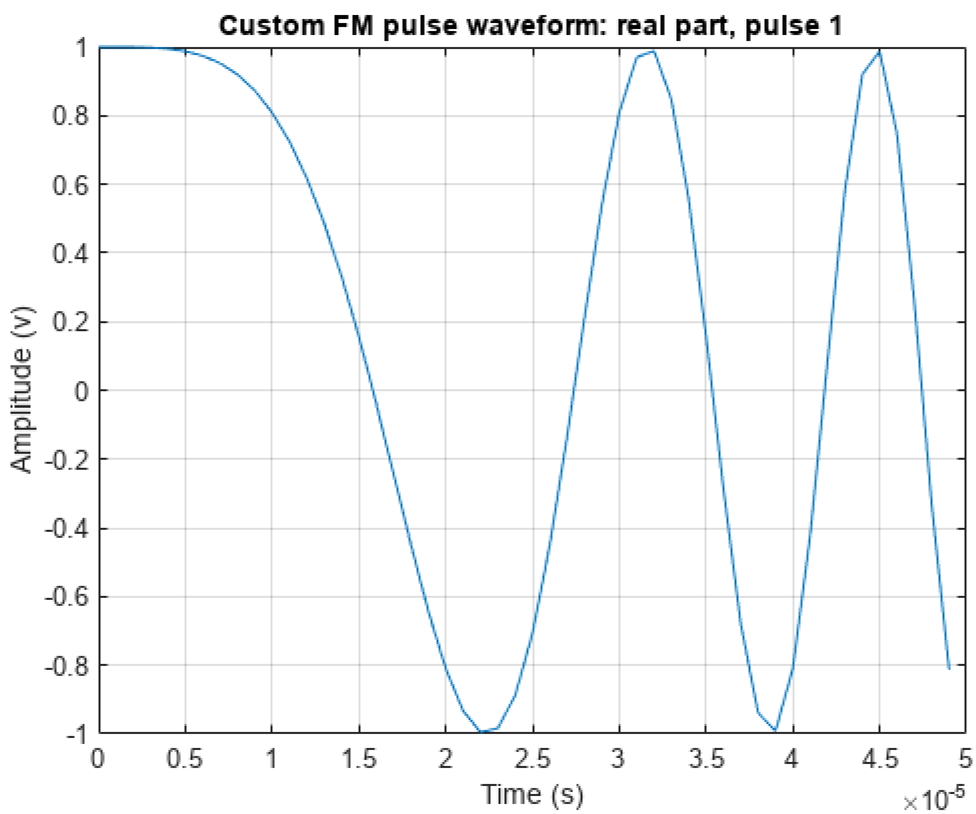
```
waveform = phased.CustomFMWaveform()
```

```
waveform = 
  phased.CustomFMWaveform with properties:

                SampleRate: 1000000
     DurationSpecification: 'Pulse width'
                PulseWidth: 5.0000e-05
                       PRF: 10000
     PRFSelectionInputPort: false
```

```
      FrequencyModulation: [0 100000]
                 Envelope: 'Rectangular'
    FrequencyOffsetSource: 'Property'
          FrequencyOffset: 0
             OutputFormat: 'Pulses'
                NumPulses: 1
            PRFOutputPort: false
   CoefficientsOutputPort: false
```

Display the real part of the waveform.

```
plot(waveform)
```



Display Sinusoidal Frequency Modulated Waveform

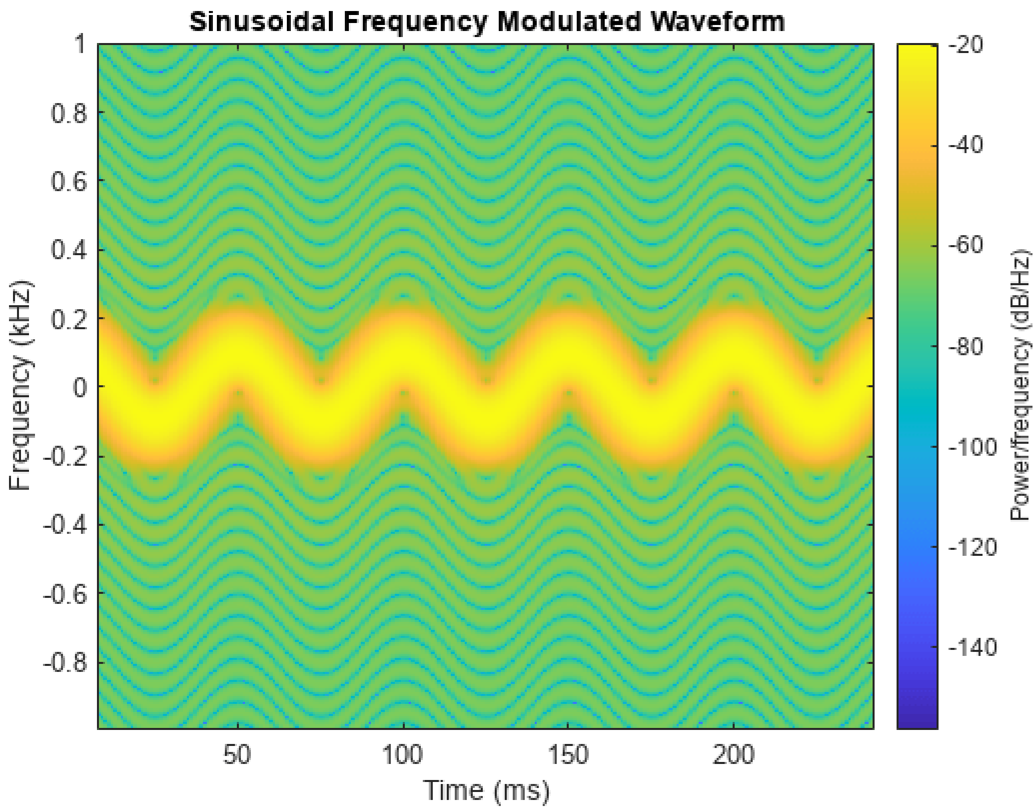
Create a sinusoidal frequency modulated waveform with a bandwidth of 200 Hz, a pulse width of 0.25 s, and a modulation frequency of 20 Hz.

```
BW = 200;
T = 0.25;
fm = 20;
fs = 10*BW;
freqfunc = @(t)(BW/2)*cos(2*pi*fm*t);
waveform = phased.CustomFMWaveform(SampleRate=fs, ...
```

```
    PulseWidth=T,FrequencyModulation=freqfunc,PRF=1/T);
wav = waveform();
```

Display the spectrogram of the waveform.

```
spectrogram(wav,32,30,512,fs,'yaxis','centered')
title('Sinusoidal Frequency Modulated Waveform')
```



Waveform Derived from Taylor Spectrum Window

Create a nonlinear FM waveform derived from a power spectral density function shaped as a Taylor window with -35 dB sidelobes. The pulse bandwidth is 120 MHz and the pulse duration is10μsec. Generate matched filter coefficients and then apply a matched filter. Plot the resulting matched filter output to display the range sidelobe levels.

```
BW = 120e6;
T = 10e-6;
fs = 10*BW;
```

Generate 200 points of a waveform with instantaneous frequency values defined by a Taylor window. The window has -35 dB sidelobe levels.

```
w = taylorwin(200,4,-35);
freq = nlfmspec2freq(BW,w);
waveform = phased.CustomFMWaveform('SampleRate',fs, ...
```

```
    'PulseWidth',T,'FrequencyModulation',freq, ...
    'OutputFormat','Pulses','CoefficientsOutputPort',true);
disp(['Bandwidth = ',num2str(bandwidth(waveform)/1e6),' MHz']);
```

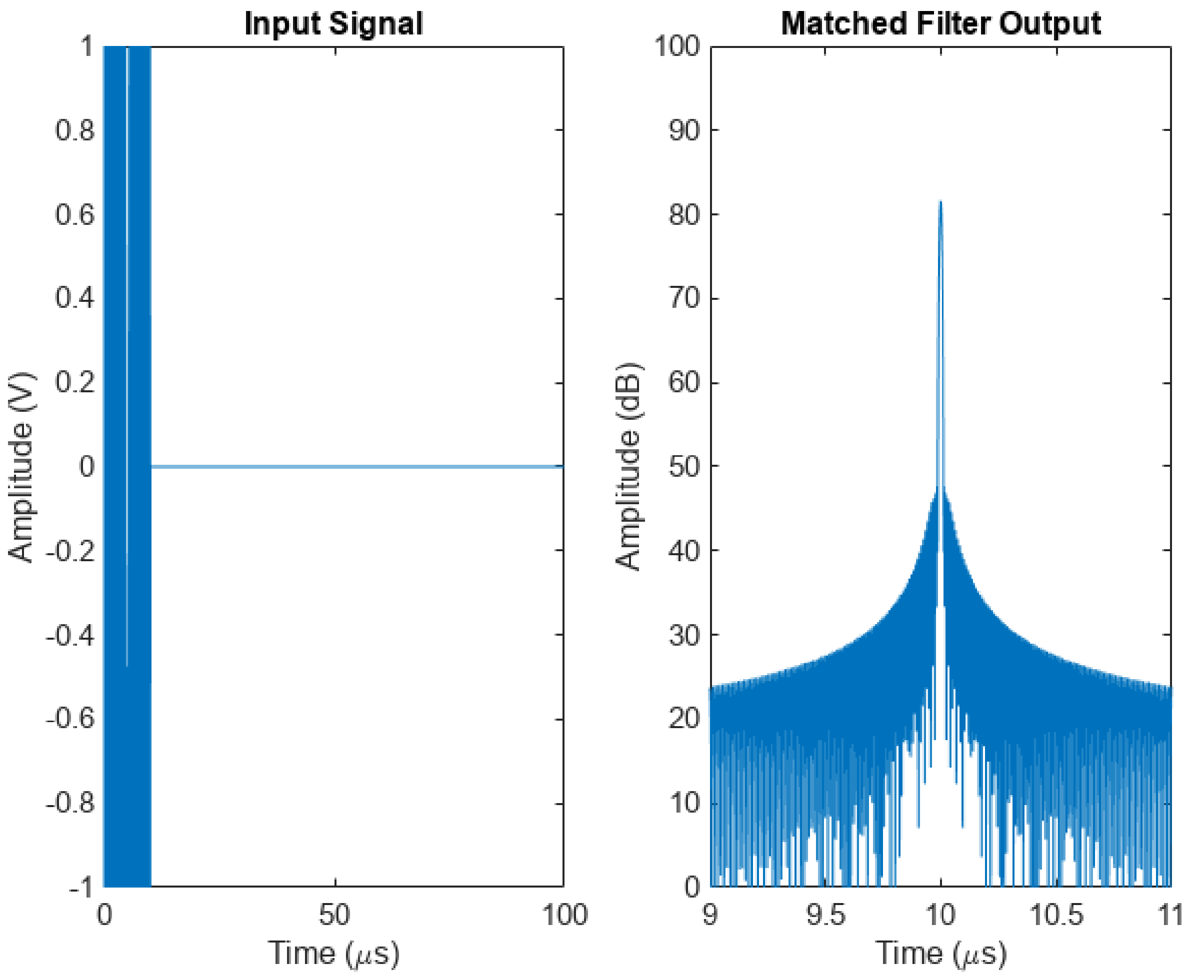
```
Bandwidth = 119.9644 MHz
```

Obtain the matched filter coefficients from the waveform.

```
[wav,coeff] = waveform();
filter = phased.MatchedFilter('CoefficientsSource','Input port');
mfout = filter(wav,coeff);
```

Plot input signal and matched filter output.

```
t = (0:numel(wav)-1)/fs;
figure
subplot(121)
plot(t*1e6,real(wav))
xlabel('Time (\mus)')
ylabel('Amplitude (V)')
title('Input Signal')

subplot(122)
plot(t*1e6,mag2db(abs(mfout)));
xlabel('Time (\mus)')
ylabel('Amplitude (dB)')
title('Matched Filter Output')
xlim([9 11]);
ylim([0 100]);
```



Version History

Introduced in R2023a

References

[1] Collins, T., and P. Atkins. "Nonlinear frequency modulation chirps for active sonar." *IEE Proceedings-Radar, Sonar and Navigation* 146.6 (1999): 312-316.

[2] Levanon, Nadav, and Eli Mozeson. *Radar signals*. John Wiley & Sons, 2004, pp. 92-93.

[3] Doerry, Armin Walter. "Generating nonlinear FM chirp waveforms for radar". No. SAND2006-5856. *Sandia National Laboratories (SNL)*, Albuquerque, NM, and Livermore, CA (United States), 2006.

[4] Cook, C. E. "A class of nonlinear FM pulse compression signals." *Proceedings of the IEEE* 52.11 (1964): 1369-1371.

[5] Yang, J., and T. K. Sarkar. "Doppler-invariant property of hyperbolic frequency modulated waveforms." *Microwave and optical technology letters* 48.6 (2006): 1174-1179.

[6] Melvin, William L., and James Scheer. *Principles of modern radar: advanced techniques.* SciTech Pub., 2013.

[7] Alphonse, Sebastian, and Geoffrey A. Williamson. "Evaluation of a class of NLFM radar signals." *EURASIP Journal on Advances in Signal Processing* 2019.1 (2019): 1-12.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.LinearFMWaveform` | `phased.NonlinearFMWaveform` | `phased.PhaseCodedWaveform` | `phased.RectangularWaveform` | `phased.SteppedFMWaveform` | `nlfmspec2freq`

Topics
"Waveform Analysis Using the Ambiguity Function"

phased.LOSChannel

Package: `phased`

Narrowband LOS propagation channel

Description

The `phased.LOSChannel` models the propagation of narrowband electromagnetic signals through a line-of-sight (LOS) channel from a source to a destination. In an LOS channel, propagation paths are straight lines from point to point. The propagation model in the LOS channel includes free-space attenuation in addition to attenuation due to atmospheric gases, rain, fog, and clouds. You can use `phased.LOSChannel` to model the propagation of signals between multiple points simultaneously.

While the System object works for all frequencies, the attenuation models for atmospheric gases and rain are valid for electromagnetic signals in the frequency range 1–1000 GHz only. The attenuation model for fog and clouds is valid for 10–1000 GHz. Outside these frequency ranges, the System object uses the nearest valid value.

The `phased.LOSChannel` System object applies range-dependent time delays to the signals, as well as gains or losses. When either the source or destination is moving, the System object applies Doppler shifts.

Like the `phased.FreeSpace` System object, the `phased.LOSChannel` System object supports two-way propagation.

To compute the propagation delay for specified source and receiver points:

1. Define and set up your LOS channel using the “Construction” on page 1-936 procedure. You can set the System object properties during construction or leave them at their default values. Some properties are tunable and can be changed at any time.
2. Call the `step` method to compute the propagated signal using the properties of the `phased.LOSChannel` System object.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`sLOS = phased.LOSChannel` creates an LOS attenuating propagation channel System object, `sLOS`.

`sLOS = phased.LOSChannel(Name,Value)` creates a System object, `sLOS`, with each specified property `Name` set to the specified `Value`. You can specify additional name and value pair arguments in any order as (`Name1,Value1`,...,`NameN,ValueN`).

Properties

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `double`

`SpecifyAtmosphere` — Enable atmospheric attenuation model
`false` (default) | `true`

Option to enable the atmospheric attenuation model, specified as a `false` or `true`. Set this property to `true` to add signal attenuation caused by atmospheric gases, rain, fog, or clouds. Set this property to `false` to ignore atmospheric effects in propagation.

Setting `SpecifyAtmosphere` to `true`, enables the `Temperature`, `DryAirPressure`, `WaterVapourDensity`, `LiquidWaterDensity`, and `RainRate` properties.

Data Types: `logical`

`Temperature` — Ambient temperature
`15` (default) | real-valued scalar

Ambient temperature, specified as a real-valued scalar. Units are in degrees Celsius.

Example: `20.0`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`DryAirPressure` — Atmospheric dry air pressure
`101.325e3` (default) | positive real-valued scalar

Atmospheric dry air pressure, specified as a positive real-valued scalar. Units are in pascals (Pa). The default value of this property corresponds to one standard atmosphere.

Example: `101.0e3`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`WaterVapourDensity` — Atmospheric water vapor density
`7.5` (default) | positive real-valued scalar

Atmospheric water vapor density, specified as a positive real-valued scalar. Units are in g/m^3.

Example: `7.4`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`LiquidWaterDensity` — Liquid water density
`0.0` (default) | nonnegative real-valued scalar

Liquid water density of fog or clouds, specified as a nonnegative real-valued scalar. Units are in g/m^3. Typical values for liquid water density are 0.05 for medium fog and 0.5 for thick fog.

Example: `0.1`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`RainRate` — Rainfall rate
`0.0` (default) | nonnegative scalar

Rainfall rate, specified as a nonnegative scalar. Units are in mm/hr.

Example: `10.0`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`TwoWayPropagation` — Enable two-way propagation
`false` (default) | `true`

Enable two-way propagation, specified as a `false` or `true`. Set this property to `true` to perform round-trip propagation between the signal origin and destination specified in `step`. Set this property to `false` to perform only one-way propagation from the origin to the destination.

Example: `true`

Data Types: `logical`

`SampleRate` — Sample rate of signal
`1e6` (default) | positive scalar

Sample rate of signal, specified as a positive scalar. Units are in Hz. The System object uses this quantity to calculate the propagation delay in units of samples.

Example: `1e6`

Data Types: `double`

`MaximumDistanceSource` — Source of maximum one-way propagation distance
`'Auto'` (default) | `'Property'`

Source of maximum one-way propagation distance, specified as `'Auto'` or `'Property'`. The maximum one-way propagation distance is used to allocate sufficient memory for signal delay computation. When you set this property to `'Auto'`, the System object automatically allocates memory. When you set this property to `'Property'`, you specify the maximum one-way propagation distance using the value of the `MaximumDistance` property.

Data Types: `char`

`MaximumDistance` — Maximum one-way propagation distance
`10000` (default) | positive real-valued scalar

Maximum one-way propagation distance, specified as a positive real-valued scalar. Units are in meters. Any signal that propagates more than the maximum one-way distance is ignored. The maximum distance must be greater than or equal to the largest position-to-position distance.

Example: `5000`

Dependencies

To enable this property, set the `MaximumDistanceSource` property to `'Property'`.

Data Types: `double`

Methods

reset	Reset states of System object
step	Propagate signal in LOS channel

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Propagate Polarized Wave in LOS Channel

Propagate a polarized electromagnetic wave radiating from a short-dipole antenna element. The dipole is rotated 30° around the *y*-axis. Set the orientation of the local axis to coincide with the dipole. Assume the dipole radiates at 30.0 GHz. Propagate the signal toward a target approximately 10 km away.

Create the short-dipole antenna element and radiator System objects. Set the `Polarization` property to `'Combined'` to generate polarized waves.

```
freq = 30.0e9;
c = physconst('LightSpeed');
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6 40e9], ...
    'AxisDirection','Z');
radiator = phased.Radiator('Sensor',antenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',freq, ...
```

```
    'Polarization','Combined', ...
    'WeightsInputPort',false);
```

Create a signal to radiate. The signal envelope consists of several cycles of a 4 kHz sinusoid with amplitude set to unity. Set the sampling frequency to 1 MHz.

```
fsig = 4.0e3;
fs = 1.0e6;
t = [1:1000]/fs;
signal = sin(2*pi*fsig*t');
laxes = roty(30)*eye(3,3);
```

Use a `phased.FreeSpace` System object™ to propagate the field from the origin to the destination in free space.

```
fschannel = phased.FreeSpace('PropagationSpeed',c,...
    'OperatingFrequency',freq,...
    'TwoWayPropagation',false,...
    'SampleRate',fs);
```

Use a `phased.LOSChannel` System object to propagate the field from the origin to the destination in the LOS channel. Attenuation is due to atmospheric gases and fog.

```
loschannel = phased.LOSChannel('PropagationSpeed',c,...
    'OperatingFrequency',freq,...
    'TwoWayPropagation',false,...
    'SampleRate',fs,'SpecifyAtmosphere',true,'LiquidWaterDensity',0.5);
```

Set the signal origin, signal origin velocity, signal destination, and signal destination velocity.

```
source_pos = [0;0;0];
target_pos = [10000;200;0];
source_vel = [0;0;0];
target_vel = [0;0;0];
[~,radiatingAngles] = rangeangle(target_pos,source_pos,laxes);
```

Radiate the signal towards the target. The radiated signal is a `struct` containing the polarized field.

```
rad_sig = radiator(signal,radiatingAngles,laxes);
```

Propagate the signals to the target in free space.

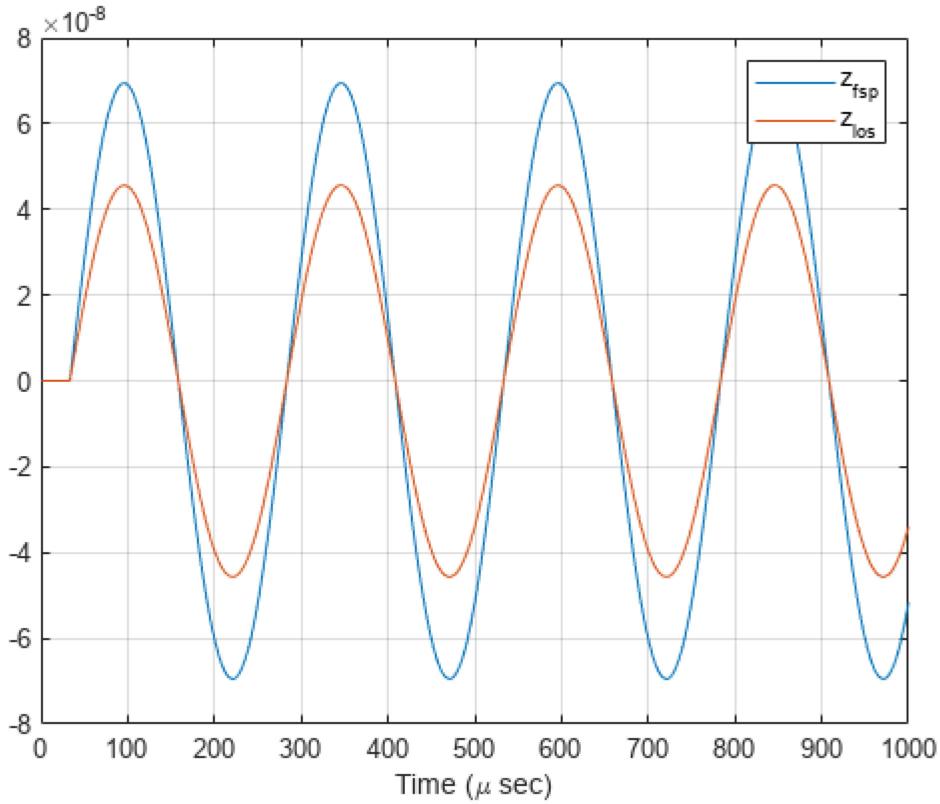
```
prop_sig = fschannel(rad_sig,source_pos,target_pos,...
    source_vel,target_vel);
```

Propagate the signals to the target in the LOS channel.

```
prop_att_sig = loschannel(rad_sig,source_pos,target_pos,...
    source_vel,target_vel);
```

Plot the z-components of both the free-space and LOS-channel-propagated signals.

```
plot(1e6*t,real(prop_sig.Z),1e6*t,real(prop_att_sig.Z))
grid
xlabel('Time (\mu sec)')
legend('z_{fsp}','z_{los}')
```



The LOS channel signal is attenuated as compared to the free-space signal.

More About

Path Attenuation or Loss

Attenuation or path loss in the LOS channel consists of four components. $L = L_{fsp}L_gL_cL_r$, where

- L_{fsp} is the free space path attenuation
- L_g is the atmospheric path attenuation
- L_c is the fog and cloud path attenuation
- L_r is the rain path attenuation

Each path attenuation is in magnitude units, not in dB.

Free-space Time Delay and Path Loss

When the origin and destination are stationary relative to each other, you can write the output signal of a free-space channel as $Y(t) = x(t-\tau)/L_{fsp}$. The quantity τ is the signal delay and L_{fsp} is the free-space path loss. The delay τ is given by R/c, where R is the propagation distance and c is the propagation speed. The free-space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far field of the transmitting element or array. In the near field, the free-space path loss formula is not valid and can result in a loss smaller than one, equivalent to a signal gain. Therefore, the loss is set to unity for range values, $R \le \lambda/4\pi$.

When the origin and destination have relative motion, the processing also introduces a Doppler frequency shift. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The quantity v is the relative speed of the destination with respect to the origin.

For more details on free-space channel propagation, see [5].

Atmospheric Gas Attenuation Model

This model calculates the attenuation of signals that propagate through atmospheric gases.

Electromagnetic signals attenuate when they propagate through the atmosphere. This effect is due primarily to the absorption resonance lines of oxygen and water vapor, with smaller contributions coming from nitrogen gas. The model also includes a continuous absorption spectrum below 10 GHz. The ITU model *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* is used. The model computes the specific attenuation (attenuation per kilometer) as a function of temperature, pressure, water vapor density, and signal frequency. The atmospheric gas model is valid for frequencies from 1–1000 GHz and applies to polarized and nonpolarized fields.

The formula for specific attenuation at each frequency is

$$\gamma = \gamma_o(f) + \gamma_w(f) = 0.1820 f N''(f) .$$

The quantity $N''()$ is the imaginary part of the complex atmospheric refractivity and consists of a spectral line component and a continuous component:

$$N''(f) = \sum_i S_i F_i + N''_D(f)$$

The spectral component consists of a sum of discrete spectrum terms composed of a localized frequency bandwidth function, $F(f)_i$, multiplied by a spectral line strength, S_i. For atmospheric oxygen, each spectral line strength is

$$S_i = a_1 \times 10^{-7} \left(\frac{300}{T}\right)^3 \exp\left[a_2(1 - \left(\frac{300}{T}\right)\right] P .$$

For atmospheric water vapor, each spectral line strength is

$$S_i = b_1 \times 10^{-1} \left(\frac{300}{T}\right)^{3.5} \exp\left[b_2(1 - \left(\frac{300}{T}\right)\right] W .$$

P is the dry air pressure, W is the water vapor partial pressure, and T is the ambient temperature. Pressure units are in hectoPascals (hPa) and temperature is in degrees Kelvin. The water vapor partial pressure, W, is related to the water vapor density, ρ, by

$$W = \frac{\rho T}{216.7} .$$

The total atmospheric pressure is $P + W$.

For each oxygen line, S_i depends on two parameters, a_1 and a_2. Similarly, each water vapor line depends on two parameters, b_1 and b_2. The ITU documentation cited at the end of this section contains tabulations of these parameters as functions of frequency.

The localized frequency bandwidth functions $F_i(f)$ are complicated functions of frequency described in the ITU references cited below. The functions depend on empirical model parameters that are also tabulated in the reference.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length, R. Then, the total attenuation is $L_g = R(\gamma_o + \gamma_w)$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Fog and Cloud Attenuation Model

This model calculates the attenuation of signals that propagate through fog or clouds.

Fog and cloud attenuation are the same atmospheric phenomenon. The ITU model, *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog* is used. The model computes the specific attenuation (attenuation per kilometer), of a signal as a function of liquid water density, signal frequency, and temperature. The model applies to polarized and nonpolarized fields. The formula for specific attenuation at each frequency is

$$\gamma_c = K_l(f)M,$$

where M is the liquid water density in gm/m^3. The quantity $K_l(f)$ is the specific attenuation coefficient and depends on frequency. The cloud and fog attenuation model is valid for frequencies 10–1000 GHz. Units for the specific attenuation coefficient are (dB/km)/(g/m^3).

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length R. Total attenuation is $L_c = R\gamma_c$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply narrowband attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Rainfall Attenuation Model

This model calculates the attenuation of signals that propagate through regions of rainfall. Rain attenuation is a dominant fading mechanism and can vary from location-to-location and from year-to-year.

Electromagnetic signals are attenuated when propagating through a region of rainfall. Rainfall attenuation is computed according to the ITU rainfall model *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. The model computes the specific attenuation (attenuation per kilometer) of a signal as a function of rainfall rate, signal frequency, polarization, and path elevation angle. The specific attenuation, γ_R, is modeled as a power law with respect to rain rate

$$\gamma_R = kR^\alpha,$$

where R is rain rate. Units are in mm/hr. The parameter k and exponent α depend on the frequency, the polarization state, and the elevation angle of the signal path. The specific attenuation model is valid for frequencies from 1–1000 GHz.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the an effective propagation distance, d_{eff}. Then, the total attenuation is $L = d_{\mathrm{eff}}\gamma_{\mathrm{R}}$.

The effective distance is the geometric distance, d, multiplied by a scale factor

$$r = \frac{1}{0.477d^{0.633}R_{0.01}^{0.073\alpha}f^{0.123} - 10.579(1 - \exp(-0.024d))}$$

where f is the frequency. The article *Recommendation ITU-R P.530-17 (12/2017): Propagation data and prediction methods required for the design of terrestrial line-of-sight systems* presents a complete discussion for computing attenuation.

The rain rate, R, used in these computations is the long-term statistical rain rate, $R_{0.01}$. This is the rain rate that is exceeded 0.01% of the time. The calculation of the statistical rain rate is discussed in *Recommendation ITU-R P.837-7 (06/2017): Characteristics of precipitation for propagation modelling*. This article also explains how to compute the attenuation for other percentages from the 0.01% value.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Version History

Introduced in R2016a

References

[1] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases*. 2013.

[2] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog*. 2013.

[3] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. 2005.

[4] Seybold, J. *Introduction to RF Propagation*. New York: Wiley & Sons, 2005.

[5] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Functions
`rangeangle` | `fogpl` | `gaspl` | `rainpl` | `fspl`

Objects
`phased.FreeSpace` | `phased.RadarTarget` | `phased.BackscatterRadarTarget` | `twoRayChannel` | `phased.WidebandFreeSpace` | `phased.WidebandLOSChannel`

reset

System object: `phased.LOSChannel`
Package: `phased`

Reset states of System object

Syntax

`reset(sLOS)`

Description

`reset(sLOS)` resets the internal state of the `phased.LOSChannel` System object, `sLOS`. If `SeedSource` is a property of this System object and has the value `'Property'`, then this method resets the random number generator state.

Input Arguments

`sLOS` — LOS channel
`phased.LOSChannel` System object

LOS channel, specified as a `phased.LOSChannel` System object.

Example: `phased.LOSChannel`

Version History

Introduced in R2016a

step

System object: phased.LOSChannel
Package: phased

Propagate signal in LOS channel

Syntax

```
prop_sig = step(sLOS,sig,origin_pos,dest_pos,origin_vel,dest_vel)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`prop_sig = step(sLOS,sig,origin_pos,dest_pos,origin_vel,dest_vel)` returns the resulting signal, `prop_sig`, when a narrowband signal, `sig`, propagates through a line-of-sight (LOS) channel from a source located at the `origin_pos` position to a destination at the `dest_pos` position. Only one of the `origin_pos` or `dest_pos` arguments can specify multiple positions. The other must contain a single position. The velocity of the signal origin is specified in `origin_vel` and the velocity of the signal destination is specified in `dest_vel`. The dimensions of `origin_vel` and `dest_vel` must match the dimensions of `origin_pos` and `dest_pos`, respectively.

Electromagnetic fields propagating through an LOS channel can be polarized or nonpolarized. For nonpolarized fields, the propagating signal field, `sig`, is a vector or matrix. For polarized fields, `sig` is an array of structures. The structure elements represent an electric field vector in Cartesian form.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

sLOS — LOS channel

`phased.LOSChannel` System object

LOS channel, specified as a `phased.LOSChannel` System object.

Example: `phased.LOSChannel`

sig — Narrowband signal

M-by-*N* complex-valued matrix | 1-by-*N* `struct` array containing complex-valued fields

Narrowband signal, specified as a matrix or `struct` array, depending on whether is signal or polarized or nonpolarized. The quantity *M* is the number of samples in the signal, and *N* is the number of LOS channels. Each channel corresponds to a source-destination pair.

- Narrowband nonpolarized scalar signal. Specify `sig` as an *M*-by-*N* complex-valued matrix. Each column contains one signal propagated along the line-of-sight path.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
- Narrowband polarized signal. Specify `sig` as a 1-by-*N* `struct` array containing complex-valued fields. Each `struct` represents a polarized signal propagated along the line-of-sight path. Each `struct` element contains three *M*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

`origin_pos` — Signal origins
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Origin of signals, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of LOS channels. If `origin_pos` is a column vector, it takes the form `[x;y;z]`. If `origin_pos` is a matrix, each column specifies a different signal origin and has the form `[x;y;z]`. Units are in meters.

You cannot specify both `origin_pos` and `dest_pos` as matrices. At least one must be a 3-by-1 column vector.

Example: `[1000;100;500]`

Data Types: `double`

`dest_pos` — Signal destinations
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Destination position of the signal or signals, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of LOS channels propagating from or to *N* signal origins. If `dest_pos` is a 3-by-1 column vector, it takes the form `[x;y;z]`. If `dest_pos` is a matrix, each column specifies a different signal destination and takes the form `[x;y;z]` Position units are in meters.

You cannot specify both `origin_pos` and `dest_pos` as matrices. At least one must be a 3-by-1 column vector.

Example: `[0;0;0]`

Data Types: `double`

`origin_vel` — Velocities of signal origins
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Velocity of signal origin, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The dimensions of `origin_vel` must match the dimensions of `origin_pos`. If `origin_vel` is a column vector, it takes the form `[Vx;Vy;Vz]`. If `origin_vel` is a 3-by-*N* matrix, each column specifies a different origin velocity and has the form `[Vx;Vy;Vz]`. Velocity units are in meters per second.

Example: `[10;0;5]`

Data Types: `double`

`dest_vel` — Velocities of signal destinations
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Velocity of signal destinations, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The dimensions of `dest_vel` must match the dimensions of `dest_pos`. If `dest_vel` is a column vector, it takes the form `[Vx;Vy;Vz]`. If `dest_vel` is a 3-by-*N* matrix, each column specifies a different destination velocity and has the form `[Vx;Vy;Vz]` Velocity units are in meters per second.

Example: `[0;0;0]`

Data Types: `double`

Output Arguments

`prop_sig` — Narrowband propagated signal
M-by-*N* complex-valued matrix | 1-by-*N* `struct` array containing complex-valued fields

Narrowband signal, returned as a matrix or `struct` array, depending on whether signal is polarized or nonpolarized. The quantity *M* is the number of samples in the signal and *N* is the number of narrowband LOS channels. Each channel corresponds to a source-destination pair.

- Narrowband nonpolarized scalar signal. `prop_sig` is an *M*-by-*N* complex-valued matrix.
- Narrowband polarized scalar signal. `prop_sig` is a 1-by-*N* `struct` array containing complex-valued fields. Each `struct` element contains three *M*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

The `prop_sig` output contains signal samples arriving at the signal destination within the current time frame. The current time frame is the time frame of the input signals to `step`. Whenever it takes longer than the current time frame for the signal to propagate from the origin to the destination, the output might not contain all contributions from the input of the current time frame. The remaining output appears in the next call to `step`.

Examples

Propagate Signal in LOS Channel

Propagate a sinusoidal signal in a line of sight (LOS) channel from a radar at *(1000,0,0)* meters to a target at *(10000,4000,500)* meters. Assume the signal propagates in medium fog specified by a liquid water density of 0.05 g/m^3. Assume that the radar and the target are stationary. The signal carrier frequency is 10 GHz. The signal frequency is 500 Hz and the sample rate is 8.0 kHz.

Set up the transmitted signal.

```
fs = 8.0e3;
dt = 1/fs;
fsig = 500.0;
fc = 10.0e9;
t = [0:dt:.01];
sig = sin(2*pi*fsig*t);
```

Set the liquid water density and specify the LOS channel System object™.

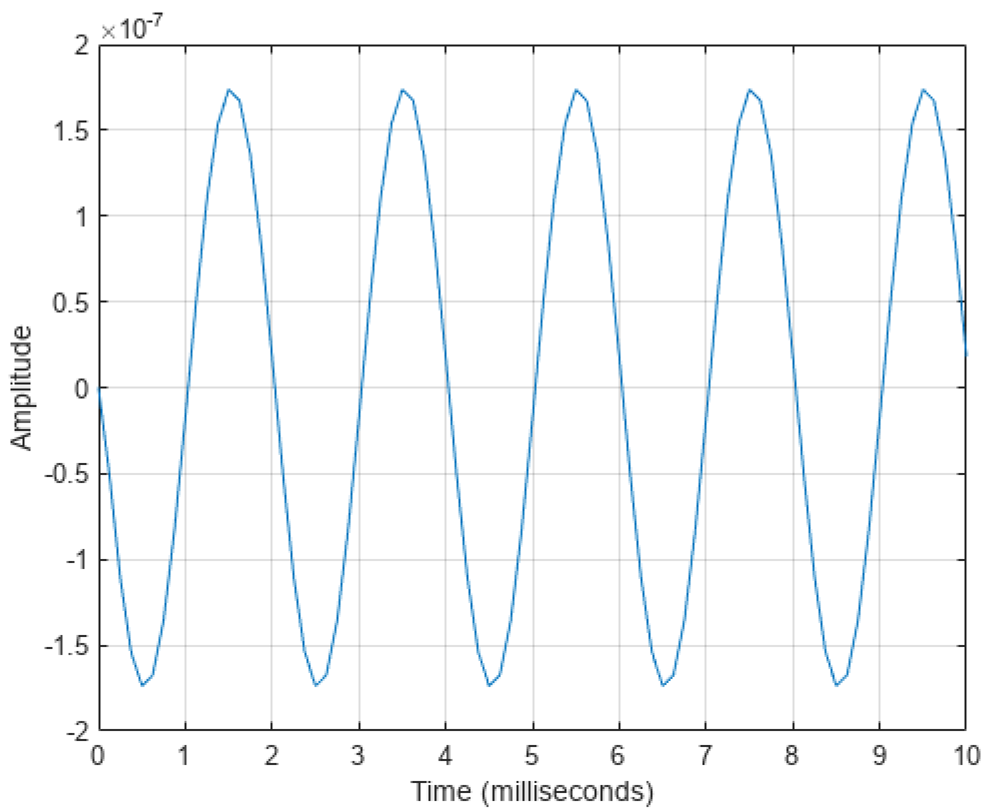
```
lwd = 0.05;
channel = phased.LOSChannel('SampleRate',fs,'SpecifyAtmosphere',true,...
    'LiquidWaterDensity',lwd,'OperatingFrequency',fc);
```

Set the origin and destination of the signal.

```
xradar = [1000,0,0].';
vradar = [0,0,0].';
xtgt = [10000,4000,500].';
vtgt = [0,0,0].';
```

Propagate the signal from origin to destination and plot the result.

```
prog_sig = channel(sig.',xradar,xtgt,vradar,vtgt);
plot(t*1000,real(prog_sig))
grid
xlabel('Time (milliseconds)')
ylabel('Amplitude')
```



Version History

Introduced in R2016a

References

[1] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases*. 2013.

[2] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog*. 2013.

[3] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. 2005.

[4] Seybold, J. *Introduction to RF Propagation*. New York: Wiley & Sons, 2005.

phased.MatchedFilter

Package: `phased`

Matched filter

Description

The `MatchedFilter` object implements matched filtering of an input signal.

To compute the matched filtered signal:

1. Define and set up your matched filter. See "Construction" on page 1-952.
2. Call `step` to perform the matched filtering according to the properties of `phased.MatchedFilter`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.MatchedFilter` creates a matched filter System object, `H`. The object performs matched filtering on the input data.

`H = phased.MatchedFilter(Name,Value)` creates a matched filter object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`CoefficientsSource`

Source of matched filter coefficients

Specify whether the matched filter coefficients come from the `Coefficients` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `Coefficients` property of this object specifies the coefficients.
`'Input port'`	An input argument in each invocation of `step` specifies the coefficients.

Default: `'Property'`

`Coefficients`

Matched filter coefficients

Specify the matched filter coefficients as a column vector. This property applies when you set the `CoefficientsSource` property to `'Property'`. This property is tunable.

Default: `[1;1]`

SpectrumWindow

Window for spectrum weighting

Specify the window used for spectrum weighting using one of `'None'`, `'Hamming'`, `'Chebyshev'`, `'Hann'`, `'Kaiser'`, `'Taylor'`, or `'Custom'`. Spectrum weighting is often used with linear FM waveform to reduce the sidelobes in the time domain. The object computes the window length internally, to match the FFT length.

Default: `'None'`

CustomSpectrumWindow

User-defined window for spectrum weighting

Specify the user-defined window for spectrum weighting using a function handle or a cell array. This property applies when you set the `SpectrumWindow` property to `'Custom'`.

If `CustomSpectrumWindow` is a function handle, the specified function takes the window length as the input and generates appropriate window coefficients.

If `CustomSpectrumWindow` is a cell array, then the first cell must be a function handle. The specified function takes the window length as the first input argument, with other additional input arguments if necessary, and generates appropriate window coefficients. The remaining entries in the cell array are the additional input arguments to the function, if any.

Default: `@hamming`

SpectrumRange

Spectrum window coverage region

Specify the spectrum region on which the spectrum window is applied as a 1-by-2 vector in the form of `[StartFrequency EndFrequency]` (in hertz). This property applies when you set the `SpectrumWindow` property to a value other than `'None'`.

Note that both `StartFrequency` and `EndFrequency` are measured in baseband. That is, they are within `[-Fs/2 Fs/2]`, where `Fs` is the sample rate that you specify in the `SampleRate` property. `StartFrequency` cannot be larger than `EndFrequency`.

Default: `[0 1e5]`

SampleRate

Coefficient sample rate

Specify the matched filter coefficients sample rate (in hertz) as a positive scalar. This property applies when you set the `SpectrumWindow` property to a value other than `'None'`.

Default: `1e6`

`SidelobeAttenuation`

Window sidelobe attenuation level

Specify the sidelobe attenuation level (in decibels) of a Chebyshev or Taylor window as a positive scalar. This property applies when you set the `SpectrumWindow` property to `'Chebyshev'` or `'Taylor'`.

Default: `30`

`Beta`

Kaiser window parameter

Specify the parameter that affects the Kaiser window sidelobe attenuation as a nonnegative scalar. Please refer to `kaiser` for more details. This property applies when you set the `SpectrumWindow` property to `'Kaiser'`.

Default: `0.5`

`Nbar`

Number of nearly constant sidelobes in Taylor window

Specify the number of nearly constant level sidelobes adjacent to the mainlobe in a Taylor window as a positive integer. This property applies when you set the `SpectrumWindow` property to `'Taylor'`.

Default: `4`

`GainOutputPort`

Output gain

To obtain the matched filter gain, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the matched filter gain, set this property to `false`.

Default: `false`

Methods

step	Perform matched filtering

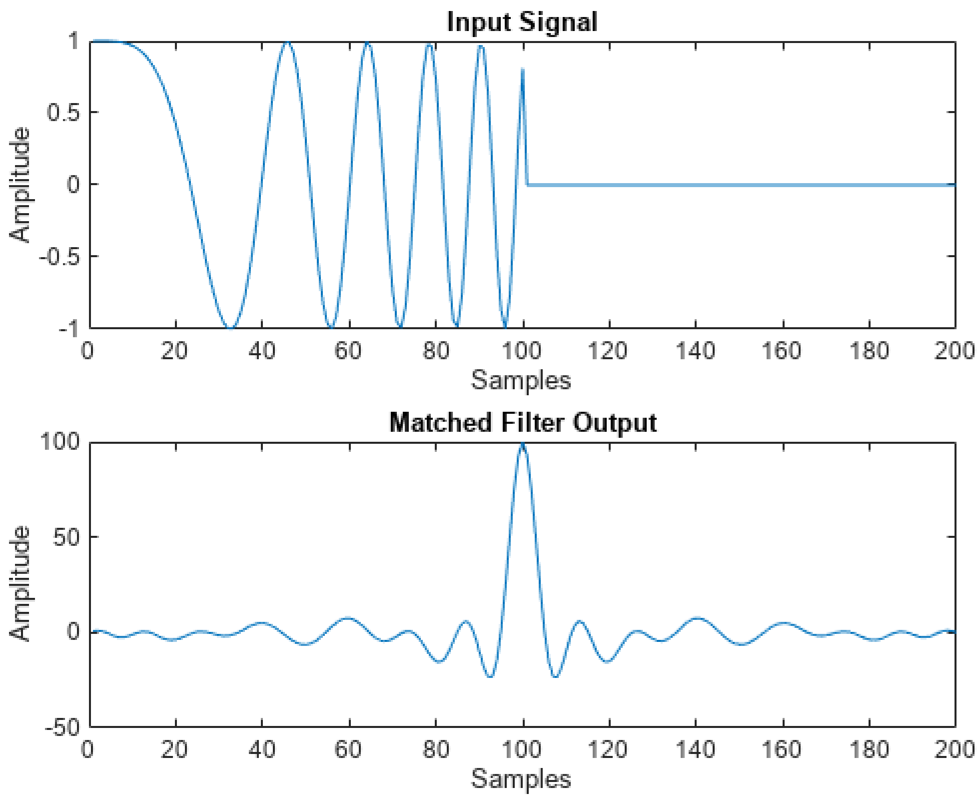
Common to All System Objects	
`release`	Allow System object property value changes

Examples

Matched Filter for Linear FM Waveform

Construct a matched filter for a linear FM waveform.

```
waveform = phased.LinearFMWaveform('PulseWidth',1e-4,'PRF',5e3);
x = waveform();
filter = phased.MatchedFilter( ...
    'Coefficients',getMatchedFilter(waveform));
y = filter(x);
subplot(2,1,1),plot(real(x))
xlabel('Samples')
ylabel('Amplitude')
title('Input Signal')
subplot(2,1,2),plot(real(y))
xlabel('Samples')
ylabel('Amplitude')
title('Matched Filter Output')
```

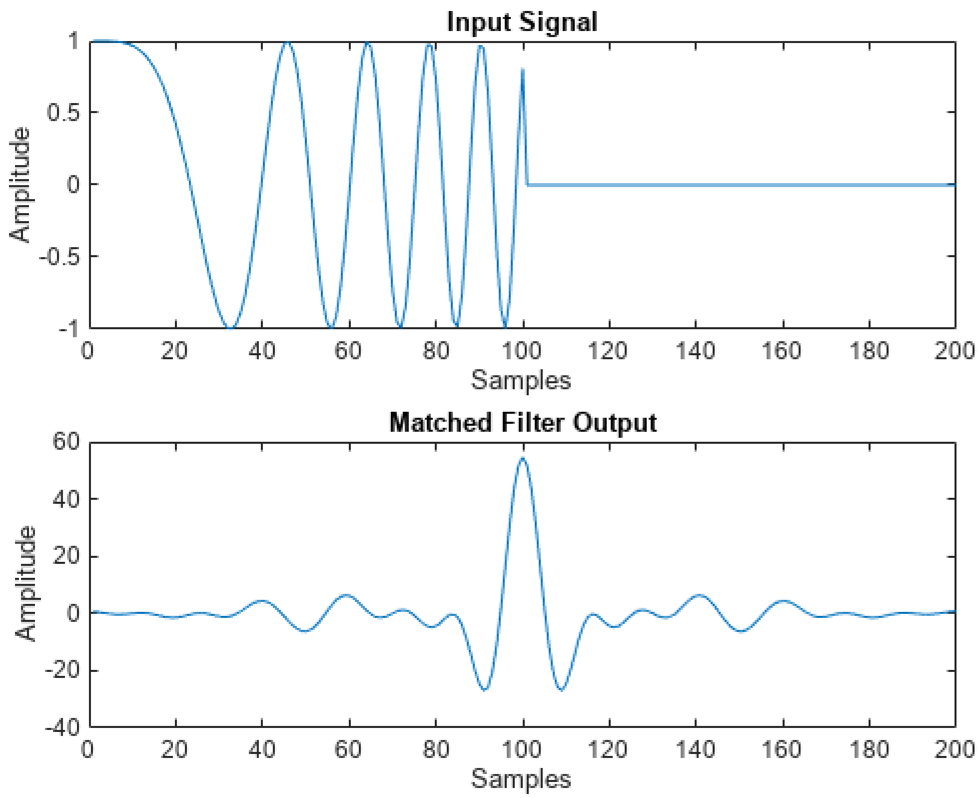


Matched Filter Using Hamming Window

Apply a matched filter, using a Hamming window to do spectrum weighting.

```
waveform = phased.LinearFMWaveform('PulseWidth',1e-4,'PRF',5e3);
x = waveform();
filter = phased.MatchedFilter( ...
    'Coefficients',getMatchedFilter(waveform), ...
    'SpectrumWindow','Hamming');
y = filter(x);
subplot(2,1,1)
```

```
plot(real(x))
xlabel('Samples')
ylabel('Amplitude')
title('Input Signal')
subplot(2,1,2)
plot(real(y))
xlabel('Samples')
ylabel('Amplitude')
title('Matched Filter Output')
```

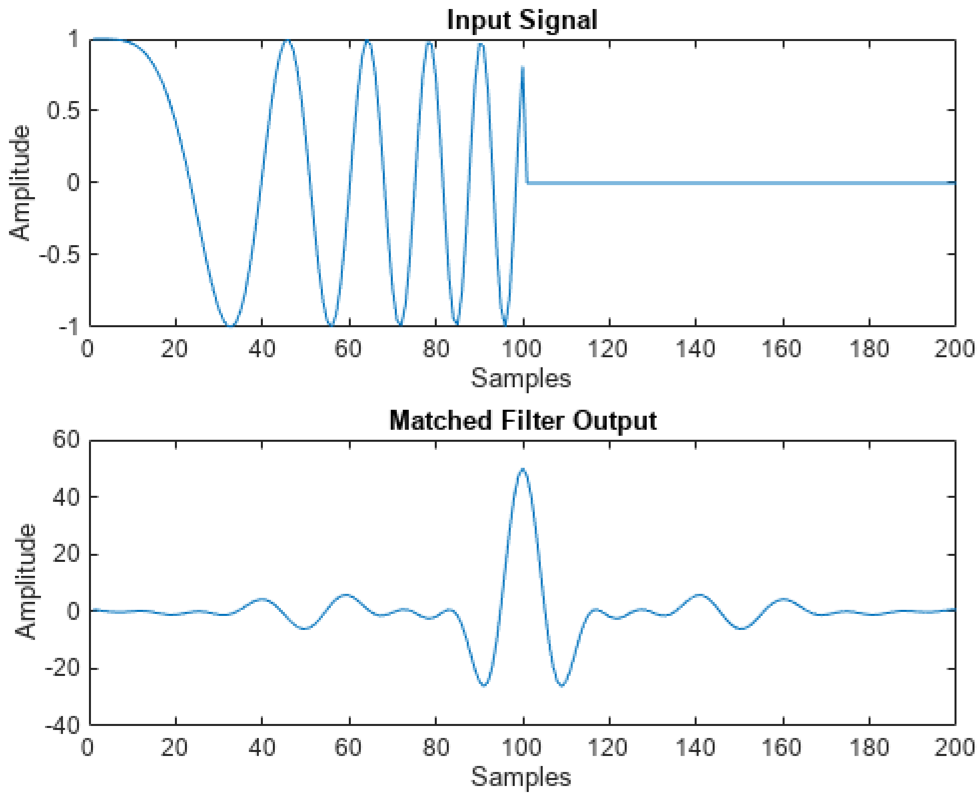


Matched Filter with Custom Window

Apply a matched filter, using a custom Gaussian window for spectrum weighting.

```
waveform = phased.LinearFMWaveform('PulseWidth',1e-4,'PRF',5e3);
x = waveform();
filter = phased.MatchedFilter( ...
    'Coefficients',getMatchedFilter(waveform), ...
    'SpectrumWindow','Custom', ...
    'CustomSpectrumWindow',{@gausswin,2.5});
y = filter(x);
subplot(2,1,1)
plot(real(x))
xlabel('Samples')
ylabel('Amplitude')
```

```
title('Input Signal')
subplot(2,1,2)
plot(real(y))
xlabel('Samples')
ylabel('Amplitude')
title('Matched Filter Output')
```



Algorithms

The filtering operation uses the overlap-add method.

Spectrum weighting produces a transfer function

$$H'(F) = w(F)H(F)$$

where $w(F)$ is the window and $H(F)$ is the original transfer function.

For further details on matched filter theory, see [1]or [2].

Version History

Introduced in R2011a

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `CustomSpectrumWindow` property is not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.CFARDetector` | `pulsint` | `phased.StretchProcessor` | `phased.TimeVaryingGain` | `taylorwin`

step

System object: `phased.MatchedFilter`
Package: `phased`

Perform matched filtering

Syntax

```
Y = step(H,X)
Y = step(H,X,COEFF)
[Y,GAIN] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` applies the matched filtering to the input `X` and returns the filtered result in `Y`. The filter is applied along the first dimension. `Y` and `X` have the same dimensions. The initial transient is removed from the filtered result.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`Y = step(H,X,COEFF)` uses the input `COEFF` as the matched filter coefficients. This syntax is available when you set the `CoefficientsSource` property to `'Input port'`.

`[Y,GAIN] = step( ___ )` returns additional output `GAIN` as the gain (in decibels) of the matched filter. This syntax is available when you set the `GainOutputPort` property to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

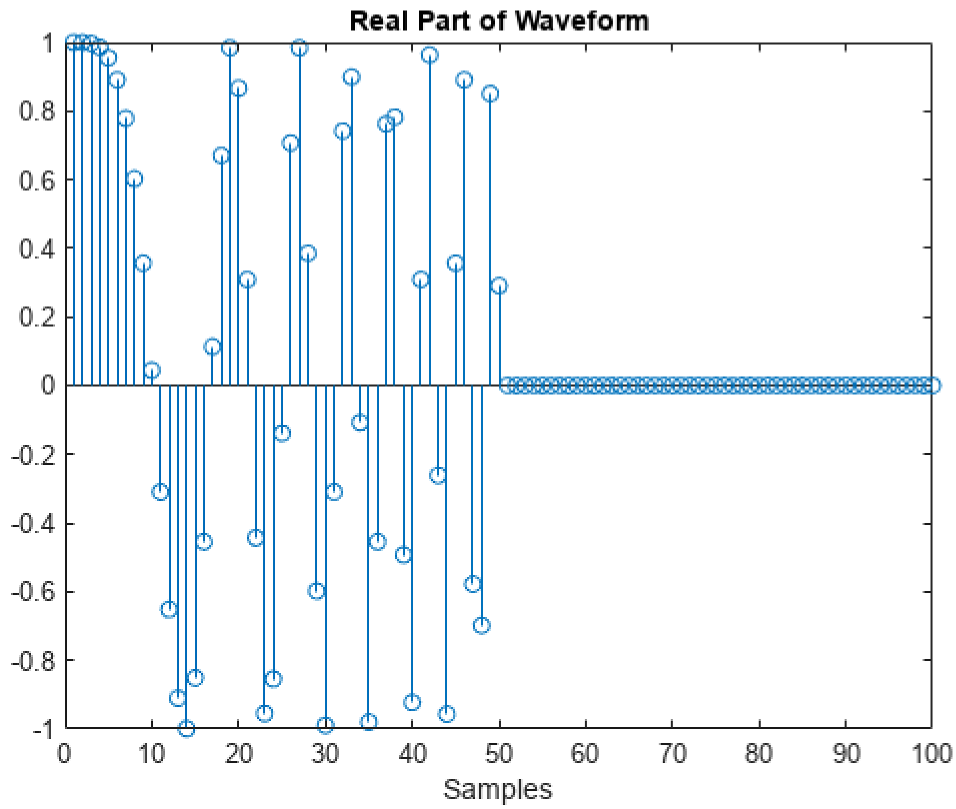
Match Filter Linear FM Waveform

Construct a linear FM waveform with a sweep bandwidth of 300 kHz and a pulse width of 50 µs. Obtain the matched filter coefficients using the `getMatchedFilter` method. Then, use the `step` to match-filter the waveform.

```
waveform = phased.LinearFMWaveform('SweepBandwidth',3e5,...
    'OutputFormat','Pulses','SampleRate',1e6,...
    'PulseWidth',50e-6,'PRF',1e4);
wav = waveform();
```

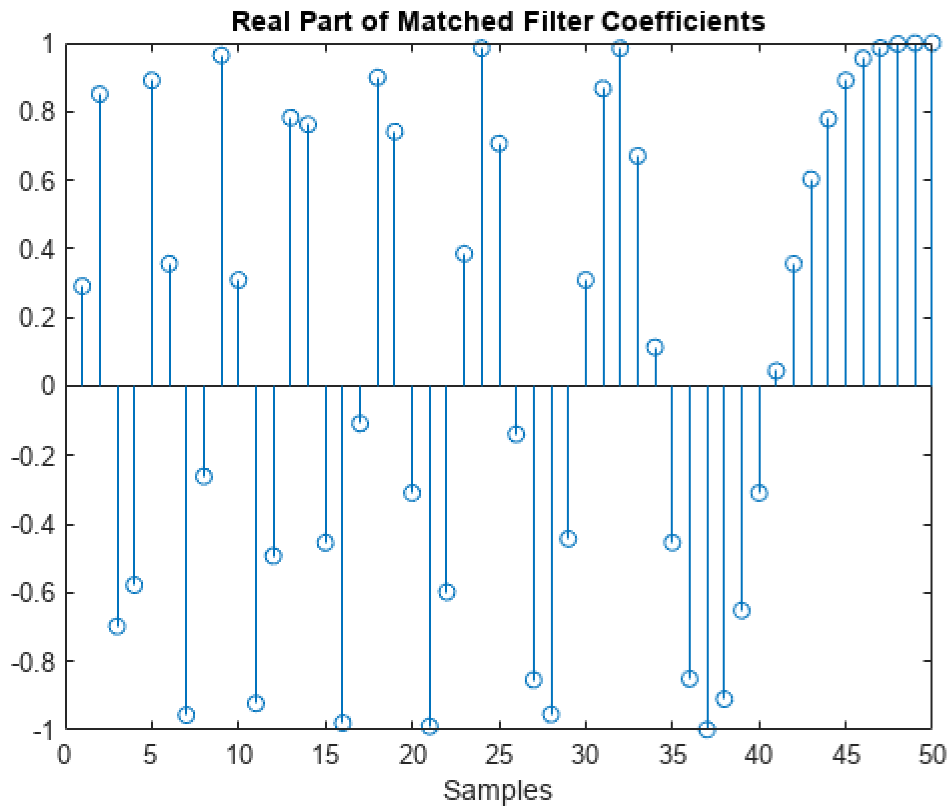
Plot the entire waveform. The length of the waveform is the pulse repetition interval (100 samples).

```
stem(real(wav))
xlabel('Samples')
title('Real Part of Waveform')
```



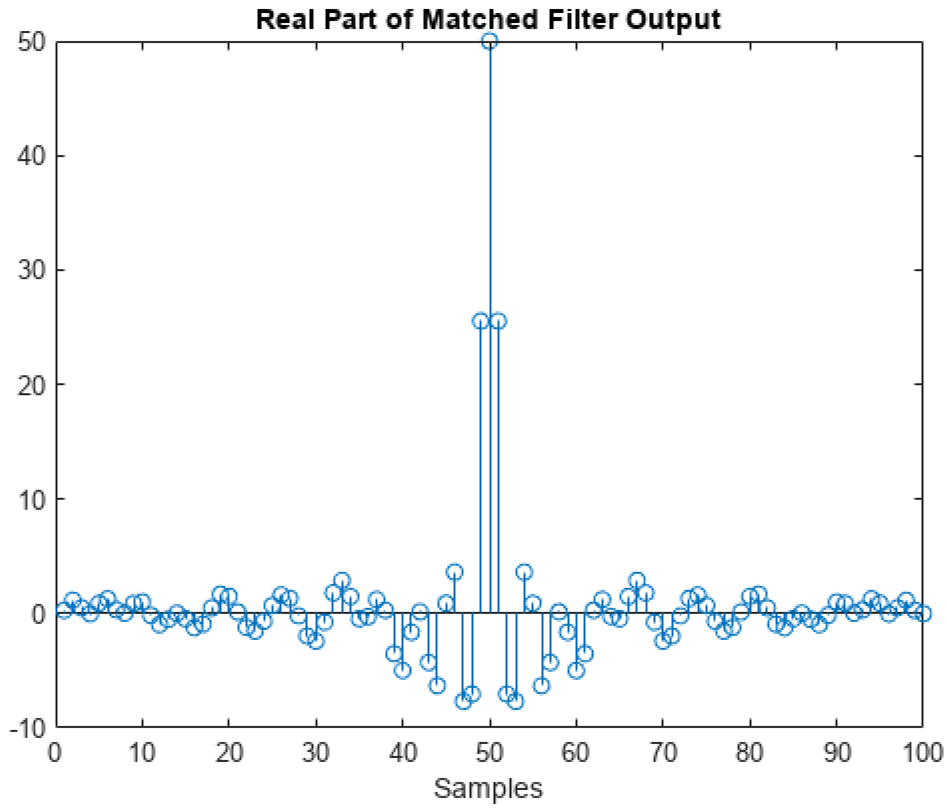
Obtain the matched filter coefficients for the linear FM waveform. The length of the matched filter coefficients is the length of the pulse.

```
mfcoeffs = getMatchedFilter(waveform);
stem(real(mfcoeffs))
xlabel('Samples')
title('Real Part of Matched Filter Coefficients')
```



Use `phased.MatchedFilter` `step` method to obtain the matched filter output.

```
filter = phased.MatchedFilter('Coefficients',mfcoeffs);
mfoutput = filter(wav);
stem(real(mfoutput))
xlabel('Samples')
title('Real Part of Matched Filter Output')
```


Real Part of Matched Filter Output
50
40
30
20
10
0
-10
0
10
20
30
40
50
60
70
80
90
100
Samples

phased.MonopulseEstimator

Package: phased

Amplitude monopulse direction finding

Description

The phased.MonopulseEstimator System object implements a target direction estimator using the amplitude monopulse technique with arbitrary array geometry. The object works with the sum and difference channels that are output from the `phased.MonopulseFeed` System object or your own sum-difference channel generator. The output is an estimate of the target direction in azimuth and elevation. You can use the object for target direction estimation and target tracking.

To create a monopulse estimator:

1. Create the `phased.MonopulseEstimator` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
estimator = phased.MonopulseEstimator
estimator = phased.MonopulseEstimator(Name,Value)
```

Description

`estimator = phased.MonopulseEstimator` creates a monopulse estimator System object, `estimator`, with default property values.

`estimator = phased.MonopulseEstimator(Name,Value)` creates an estimator with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `estimator = phased.MonopulseEstimator('SensorArray',phased.URA,'OperatingFrequency',300e6,'Coverage','Azimuth')` sets the sensor array to a uniform rectangular array (URA) with default URA property values. The estimator estimates azimuth from the sum channel and azimuth difference channel. The estimator operates at 300 MHz.

Note You can also create a `phased.MonopulseEstimator` object from a `phased.MonopulseFeed` object using the `getMonopulseEstimator` object function.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SensorArray` — Sensor array
`phased.ULA` array with default property values (default) | Phased Array System Toolbox array

Sensor array, specified as an array System object belonging to Phased Array System Toolbox. The sensor array can contain subarrays.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `double`

`Coverage` — Monopulse coverage
`'3D'` (default) | `'Azimuth'`

Coverage of monopulse estimator, specified as `'3D'` or `'Azimuth'`. When you set this property to `'3D'`, the monopulse estimator uses the sum channel and both azimuth and elevation difference channels. When you set this property to `'Azimuth'`, the estimator uses the sum channel and the azimuth difference channel.

`SquintAngle` — Squint angle
`10` (default) | scalar | real-valued 2-by-1 vector

Squint angle, specified as a scalar or real-valued 2-by-1 vector. The squint angle is the separation angle or angles between the sum beam and the beams along the azimuth and elevation directions.

- When you set the `Coverage` property to `'Azimuth'`, set the `SquintAngle` property to a scalar.
- When you set the `Coverage` property to `'3D'`, you can specify the squint angle as either a scalar or vector. If you set the `SquintAngle` property to a scalar, then the squint angle is the same along both the azimuth and elevation directions. If you set the `SquintAngle` property to a 2-by-1 vector, its elements specify the squint angle along the azimuth and elevation directions.

Example: `[20;5]`

`OutputFormat` — Output direction format
`'Angle'` (default) | `'Angle offset'`

Format of direction output, specified as `'Angle'` or `'Angle offset'`. When you set this property to `'Angle'`, the output angles are in the direction of the target. When you set this property to `'Angle offset'`, the output is the angle offset from the array steering direction.

`SumDifferenceRatioOutputPort` — Enable sum-difference ratio output
`false` (default) | `true`

Set this property to `true` to output the ratio of the sum and difference channels in the azimuth and elevation directions. Set this property to `false` to not output the ratios. The ratio is often used as an error control signal.

Data Types: `logical`

Usage

Syntax

```
angest = estimator(sumchan,diffazchan,steervec)
angest = estimator(sumchan,diffazchan,diffelchan,steervec)
[angest,dratio] = estimator(___)
```

Description

`angest = estimator(sumchan,diffazchan,steervec)` returns the estimated target angle, `angest`, derived from the sum channel signal, `sumchan`, and the azimuth difference channel signal, `diffazchan`. `steervec` specifies the array steering direction. To use this syntax, set the `Coverage` property to `'Azimuth'`.

`angest = estimator(sumchan,diffazchan,diffelchan,steervec)` also specifies the elevation difference channel signal, `diffelchan`. To use this syntax, set the `Coverage` property to `'3D'`.

`[angest,dratio] = estimator(___)` also returns the sum and difference ratio, `dratio`. To use this syntax, set the `SumDifferenceRatioOutputPort` property to `true`.

You can combine optional input arguments when their enabling properties are set. Optional inputs must be listed in the same order as the order of the enabling properties. For example:

```
[angest,dratio] = estimator(X,steervec)
```

Input Arguments

`sumchan` — Sum-channel signal
complex-valued *N*-by-1 column vector

Sum-channel signal, specified as a complex-valued *N*-by-1 column vector. *N* is the number of snapshots in the signal.

Data Types: `double`
Complex Number Support: Yes

diffazchan — Azimuth difference-channel signal
complex-valued *N*-by-1 column vector

Azimuth difference-channel signal, specified as a complex-valued *N*-by-1 column vector. *N* is the number of snapshots in the signal.

Data Types: double
Complex Number Support: Yes

diffelchan — Elevation difference-channel signal
complex-valued *N*-by-1 column vector

Elevation difference-channel signal, specified as a complex-valued *N*-by-1 column vector. *N* is the number of snapshots in the signal.

Dependencies

To enable this output argument, set the Coverage property to '3D'.

Data Types: double
Complex Number Support: Yes

steervec — Array steering direction
scalar | real-valued 2-by-1 column vector

Array steering direction, specified as a scalar or real-valued 2-by-1 column vector.

- When you set the Coverage property to 'Azimuth', the steering direction is a scalar and represents the azimuth steering angle.
- When you set the Coverage property to '3D', the steering vector has the form [azimuthAngle; elevationAngle], where azimuthAngle is the azimuth steering angle and elevationAngle is the elevation steering angle.

Units are in degrees. Azimuth angles lie between –180° and 180°, inclusive and elevation angles must lie between –90° and 90°, inclusive.

Example: [40;10]

Data Types: double

Output Arguments

angest — Estimated direction of target
real-valued 1-by-*N* vector | real-valued 2-by-*N* matrix

Estimated direction of target, returned as a real-valued 1-by-*N* vector or real-valued 2-by-*N* matrix. *N* is the number of snapshots in the signal. Units are in degrees.

- When you set the Coverage property to 'Azimuth', angest is a real-valued 1-by-*N* vector. The elements contain the estimated target direction azimuth angle at each signal snapshot.
- When you set the Coverage property to '3D', angest is a real-valued 2-by-*N* matrix. Each column contains the estimated target direction in the form [azimuthAngle;elevationAngle], where azimuthAngle is the estimated azimuth angle, and elevationAngle is the estimated elevation angle.

If you set the OutputFormat property to 'Angle offset', each element of the vector or matrix represent an offset from the steering vector direction.

Data Types: double

dratio — Ratio of sum and difference channels
real-valued 1-by-*N* vector | real-valued 2-by-*N* matrix

Ratio of sum and difference channels, returned as a real-valued 1-by-*N* vector or real-valued 2-by-*N* matrix. *N* is the number of snapshots in the signal. Units are in degrees.

- When you set the `Coverage` property to `'Azimuth'`, `dratio` is a real-valued 1-by-*N* vector. The elements contain the ratio of the sum to azimuth difference channel at each signal snapshot.
- When you set the `Coverage` property to `'3D'`, `dratio` is a real-valued 2-by-*N* matrix. The elements of the first row contain the ratio of the sum to azimuth difference channel at each signal snapshot. The elements of the second row contain the ratio of the sum to elevation difference channel at each signal snapshot.

Dependencies

To enable this output argument, set the `SumDifferenceRatioOutputPort` property to `true`.

Data Types: double

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Create Sum and Difference Channels for URA

After creating sum and difference channels, determine the direction of a target at approximately 24 degrees azimuth and 40 degrees elevation with respect to a 5-by-5 uniform rectangular array.

Create a monopulse feed system based on a URA.

```
fc = 200e6;
c = physconst('LightSpeed');
lambda = c/fc;
array = phased.URA('Size',[5 5],'ElementSpacing',lambda/2);
feed = phased.MonopulseFeed('SensorArray',array,'OperatingFrequency', ...
    fc,'Coverage','3D','AngleOutputPort',true);
```

Create a signal using a steering vector.

```
steervector = phased.SteeringVector('SensorArray',array);
x = steervector(feed.OperatingFrequency,[24;40]).';
```

Obtain the sum and difference channels and the estimated target angle.

```
[sumch,azch,elch,est_dir] = feed(x,[30;35]);
disp(est_dir)
```

```
   24.3705
   41.1997
```

Use a derived `phased.MonopulseEstimator` object to also obtain the target angle.

```
estimator = getMonopulseEstimator(feed);
est_dir = estimator(sumch,azch,elch,[30;35])
```

```
est_dir = 2×1

   24.3705
   41.1997
```

Find Direction of Target

Determine the direction of a target using monopulse processing of signals arriving on a URA. The target echo is first detected before applying monopulse processing.

```
array = phased.URA('Size',4);
collect = phased.Collector('Sensor',array);
feed = phased.MonopulseFeed('SensorArray',array,'Coverage','3D');
estimator = phased.MonopulseEstimator('SensorArray',array,'Coverage','3D');

% Create a 100-sample random source signal with a single spike to simulate
% an echo.
x = sqrt(0.01/2)*(randn(100,1)+1i*randn(100,1));
x(20) = 1;
targetangle = [31;9];
rx = collect(x,targetangle);
```

Point the monopulse in a different direction from the target. Then, create the sum and difference angles.

```
steerangle = [30;10];
[sumch,azch,elch] = feed(rx,steerangle);

% Detect the target by finding the peak of the sum channel.
[~,idx] = max(abs(sumch));

% Estimate the arrival angle using a monopulse estimator.
est_dir = estimator(sumch(idx),azch(idx),elch(idx),steerangle)
```

```
est_dir = 2×1

   31.1307
    9.0132
```

Version History

Introduced in R2018b

References

[1] Mahafza, B.R. *Radar System Analysis and Design Using Matlab*. Boca Raton: Chapman and Hall/CRC, 2000.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects
`phased.SumDifferenceMonopulseTracker` | `phased.SumDifferenceMonopulseTracker2D` | `phased.MonopulseFeed`

Functions
`getMonopulseEstimator`

phased.MonopulseFeed

Package: `phased`

Creates sum and difference channels

Description

The phased.MonopulseFeed System object implements a monopulse feed system for the amplitude sum and difference monopulse tracker. This object combines received signals from an arbitrary array to form sum and difference channels. You can use this object as a feed for the `phased.MonopulseEstimator` System object.

To create a monopulse feed system:

1. Create the `phased.MonopulseFeed` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
feed = phased.MonopulseFeed
feed = phased.MonopulseFeed(Name,Value)
```

Description

`feed = phased.MonopulseFeed` creates a monopulse feed System object, `feed`, with default property values.

`feed = phased.MonopulseFeed(Name,Value)` creates a feed system with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `feed = phased.MonopulseFeed('SensorArray',phased.URA,'OperatingFrequency',300e6,'Coverage','Azimuth')` sets the sensor array to a uniform rectangular array (URA) with default URA property values. The feed forms only the sum channel and azimuth difference channel. The feed system operates at 300 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

SensorArray — Sensor array
`phased.ULA` array with default property values (default) | Phased Array System Toolbox array

Sensor array, specified as an array System object belonging to Phased Array System Toolbox. The sensor array can contain subarrays.

Example: `phased.URA`

PropagationSpeed — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

OperatingFrequency — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `double`

Coverage — Monopulse coverage
`'3D'` (default) | `'Azimuth'`

Coverage directions of monopulse feed, specified as `'3D'` or `'Azimuth'`. When you set this property to `'3D'`, the monopulse feed forms the sum channel and both azimuth and elevation difference channels. When you set this property to `'Azimuth'`, the monopulse feed forms the sum channel and the azimuth difference channel.

Example: `'Azimuth'`

SquintAngle — Squint angle
`10` (default) | scalar | real-valued 2-by-1 vector

Squint angle, specified as a scalar or real-valued 2-by-1 vector. The squint angle is the separation angle or angles between the sum beam and the beams along the azimuth and elevation directions.

- When you set the `Coverage` property to `'Azimuth'`, set the `SquintAngle` property to a scalar.
- When you set the `Coverage` property to `'3D'`, you can specify the squint angle as either a scalar or vector. If you set the `SquintAngle` property to a scalar, then the squint angle is the same along both the azimuth and elevation directions. If you set the `SquintAngle` property to a 2-by-1 vector, its elements specify the squint angle along the azimuth and elevation directions.

Example: `[20;5]`

AngleOutputPort — Enable angle estimate output
`false` (default) | `true`

Enable angle estimate output, specified as `false` or `true`. Set this property to `true` to output the angle estimate in addition to sum and difference channels. Set this property to `false` to only output sum and difference channels.

Data Types: `logical`

Usage

Syntax

```
[sumchan,diffazchan] = feed(X,steervec)
[sumchan,diffazchan,diffelchan] = feed(X,steervec)
[ ___ ,angest] = feed(X,steervec)
```

Description

`[sumchan,diffazchan] = feed(X,steervec)` returns the sum channel signal, `sumchan`, and the azimuth difference channel signal, `diffazchan`, computed from the input signal, `X`. `steervec` specifies the array steering direction. To use this syntax, set the `Coverage` property to `'Azimuth'`.

`[sumchan,diffazchan,diffelchan] = feed(X,steervec)` also returns the elevation difference channel signal, `diffelchan`. To use this syntax, set the `Coverage` property to `'3D'`.

`[ ___ ,angest] = feed(X,steervec)` also returns the estimated direction angle, `angest`. To use this syntax, set the `AngleOutputPort` property to `true`.

Input Arguments

`X` — Input signal
complex-valued M-by-N matrix | -by-*N* matrix

Input signal, specified as a complex-valued *M*-by-*N* matrix, where *M* is the number of samples or snapshots of data, and *N* is the number of array elements. If the array contains subarrays, then *N* is the number of subarrays.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

`steervec` — Array steering direction
scalar | real-valued 2-by-1 column vector

Array steering direction, specified as a scalar or real-valued 2-by-1 column vector.

- When you set the `Coverage` property to `'Azimuth'`, the steering direction is a scalar and represents the azimuth steering angle.
- When you set the `Coverage` property to `'3D'`, the steering vector has the form `[azimuthAngle; elevationAngle]`, where `azimuthAngle` is the azimuth steering angle and `elevationAngle` is the elevation steering angle.

Units are in degrees. Azimuth angles lie between –180° and 180°, inclusive and elevation angles must lie between –90° and 90°, inclusive.

Example: [40;10]

Data Types: double

You can combine optional input arguments when their enabling properties are set. Optional inputs must be listed in the same order as the order of the enabling properties. For example,

```
array = phased.URA('Size',[5 5]);
feed = phased.MonopulseFeed('SensorArray',array,'Coverage','3D', ...
       'AngleOutputPort',true);
[sumch,dazch,delch,angest] = feed(X,steervec);
```

Output Arguments

sumchan — Sum-channel signal
complex-valued *M*-by-*1* column vector

Sum-channel signal, returned as a complex-valued *M*-by-1 column vector, where *M* is the number of rows of X.

Data Types: double
Complex Number Support: Yes

diffazchan — Azimuth difference-channel signal
complex-valued *M*-by-*1* column vector

Azimuth difference-channel signal, returned as a complex-valued *M*-by-1 column vector, where *M* is the number of rows of X.

Data Types: double
Complex Number Support: Yes

diffelchan — Elevation difference-channel signal
complex-valued *M*-by-*1* column vector

Elevation difference-channel signal, returned as a complex-valued *M*-by-1 column vector, where *M* is the number of rows of X.

Dependencies

To enable this output argument, set the Coverage property to '3D'.

Data Types: double
Complex Number Support: Yes

angest — Estimated direction of target
real-valued 2-by-1 vector

Estimated direction of target, returned as a real-valued 2-by-1 vector in the form [azimuth,elevation]. Units are in degrees.

Dependencies

To enable this output argument, set the AngleOutputPort property to true.

Data Types: double

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to phased.MonopulseFeed

getMonopulseEstimator	Create monopulse estimator from monopulse feed

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Create Sum and Difference Channels for URA

After creating sum and difference channels, determine the direction of a target at approximately 24 degrees azimuth and 40 degrees elevation with respect to a 5-by-5 uniform rectangular array.

Create a monopulse feed system based on a URA.

```
fc = 200e6;
c = physconst('LightSpeed');
lambda = c/fc;
array = phased.URA('Size',[5 5],'ElementSpacing',lambda/2);
feed = phased.MonopulseFeed('SensorArray',array,'OperatingFrequency', ...
    fc,'Coverage','3D','AngleOutputPort',true);
```

Create a signal using a steering vector.

```
steervector = phased.SteeringVector('SensorArray',array);
x = steervector(feed.OperatingFrequency,[24;40]).';
```

Obtain the sum and difference channels and the estimated target angle.

```
[sumch,azch,elch,est_dir] = feed(x,[30;35]);
disp(est_dir)
```

```
   24.3705
   41.1997
```

Use a derived `phased.MonopulseEstimator` object to also obtain the target angle.

```
estimator = getMonopulseEstimator(feed);
est_dir = estimator(sumch,azch,elch,[30;35])
```

```
est_dir = 2×1

   24.3705
```

```
41.1997
```

Version History

Introduced in R2018b

References

[1] Mahafza, B.R. *Radar System Analysis and Design Using Matlab*. Boca Raton: Chapman and Hall/CRC, 2000.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects

`phased.SumDifferenceMonopulseTracker` | `phased.SumDifferenceMonopulseTracker2D` | `phased.MonopulseEstimator`

getMonopulseEstimator

Package: phased

Create monopulse estimator from monopulse feed

Syntax

```
estimator = getMonopulseEstimator(feed)
```

Description

`estimator = getMonopulseEstimator(feed)` creates a `phased.MonopulseEstimator` System object, `estimator`, from a `phased.MonopulseFeed` System object, `feed`.

Examples

Create Sum and Difference Channels for URA

After creating sum and difference channels, determine the direction of a target at approximately 24 degrees azimuth and 40 degrees elevation with respect to a 5-by-5 uniform rectangular array.

Create a monopulse feed system based on a URA.

```
fc = 200e6;
c = physconst('LightSpeed');
lambda = c/fc;
array = phased.URA('Size',[5 5],'ElementSpacing',lambda/2);
feed = phased.MonopulseFeed('SensorArray',array,'OperatingFrequency', ...
    fc,'Coverage','3D','AngleOutputPort',true);
```

Create a signal using a steering vector.

```
steervector = phased.SteeringVector('SensorArray',array);
x = steervector(feed.OperatingFrequency,[24;40]).';
```

Obtain the sum and difference channels and the estimated target angle.

```
[sumch,azch,elch,est_dir] = feed(x,[30;35]);
disp(est_dir)
```

```
   24.3705
   41.1997
```

Use a derived `phased.MonopulseEstimator` object to also obtain the target angle.

```
estimator = getMonopulseEstimator(feed);
est_dir = estimator(sumch,azch,elch,[30;35])
```

```
est_dir = 2×1

   24.3705
```

```
41.1997
```

Input Arguments

`feed` — Monopulse feed
`phased.MonopulseFeed` System object

Monopulse feed, specified as a System object.

Output Arguments

`estimator` — Monopulse estimator
`phased.MonopulseEstimator` System object

Monopulse estimator, returned as a `phased.MonopulseEstimator` System object.

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

phased.MFSKWaveform

Package: `phased`

MFSK waveform

Description

The multiple frequency shift keying (MFSK) waveform is used in automotive radar to improve simultaneous range and Doppler estimation of multiple targets. The `MFSKWaveform` System object creates the baseband representation of an MFSK waveform. An MFSK waveform consists of two interleaved sequences of increasing frequencies, as described in “Algorithms” on page 1-981.

To obtain waveform samples:

1. Define and set up the MFSK waveform. See “Construction” on page 1-978.
2. Call `step` to generate the MFSK waveform samples according to the properties of `phased.MFSKWaveform`. The behavior of `step` is specific to each object in the toolbox. The output of the `step` method is controlled by the `OutputFormat` property, which has no effect on the properties of the waveform.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

Construction

`sMFSK = phased.MFSKWaveform` creates an MFSK waveform System object, `sMFSK`.

`sMFSK = phased.MFSKWaveform(Name,Value)` creates an MFSK waveform object, `sMFSK`, with additional properties specified by one or more `Name-Value pair` arguments. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,…,NameN,ValueN`.

Properties

`SampleRate` — Sample rate
1e6 (default) | positive scalar

Sample rate of the signal, specified as a positive scalar. Units are hertz.

Example: 96e6

Data Types: `double`

`SweepBandwidth` — MFSK sweep bandwidth
1e5 (default) | positive scalar

MFSK sweep bandwidth, specified as a positive scalar. Units are in hertz. The sweep bandwidth is the difference between the highest and lowest frequencies of either sequence.

Example: 9e7

Data Types: `double`

`StepTime` — Duration of frequency step
1e-4 (default) | positive scalar

Time duration of each frequency step, specified as a positive scalar in seconds.

Example: 0.2e-3

Data Types: `double`

`StepsPerSweep` — Total number of frequency steps
64 (default) | even positive integer

Total number of frequency steps in a sweep, specified as an even positive integer.

Example: 16

Data Types: `double`

`FrequencyOffset` — Chirp offset frequency
1000 (default) | real scalar

Chirp offset frequency, specified as a real scalar. Units are in hertz. The offset determines the frequency translation between the two sequences.

Example: 500

Data Types: `double`

`OutputFormat` — Output signal grouping
`'Steps'` (default) | `'Sweeps'` | `'Samples'`

Output signal grouping, specified as one of `'Steps'`, `'Sweeps'`, or `'Samples'`. This property has no effect on the waveform but determines the output form of the `step` method.

- `'Steps'` — The output consists of all samples contained in an integer number of frequency steps, `NumSteps`.
- `'Samples'` — The output consists of an integer number of samples, `NumSamples`.
- `'Sweeps'` — The output consists of all samples contained in an integer number of sweeps, `NumSweeps`.

Example: `'Samples'`

Data Types: `char`

`NumSamples` — Number of samples in output
1 (default) | positive integer

Number of samples in output, specified as a positive integer. This property applies only when you set `OutputFormat` to `'Samples'`.

Example: 200

Data Types: `double`

NumSteps — Number of frequency steps in output
1 (default) | positive integer

Number of frequency steps in output, specified as a positive integer. This property applies only when you set `OutputFormat` to `'Steps'`.

Example: 10

Data Types: `double`

NumSweeps — Number of sweeps in output
1 (default) | positive integer

Number of sweeps in output, specified as a positive integer. This property applies only when you set `OutputFormat` to `'Sweeps'`.

Example: 5

Data Types: `double`

Methods

plot	Plot continuous MFSK waveform
reset	Reset states of the MFSK waveform object
step	Samples of continuous MFSK waveform

Common to All System Objects	
`release`	Allow System object property value changes

Examples

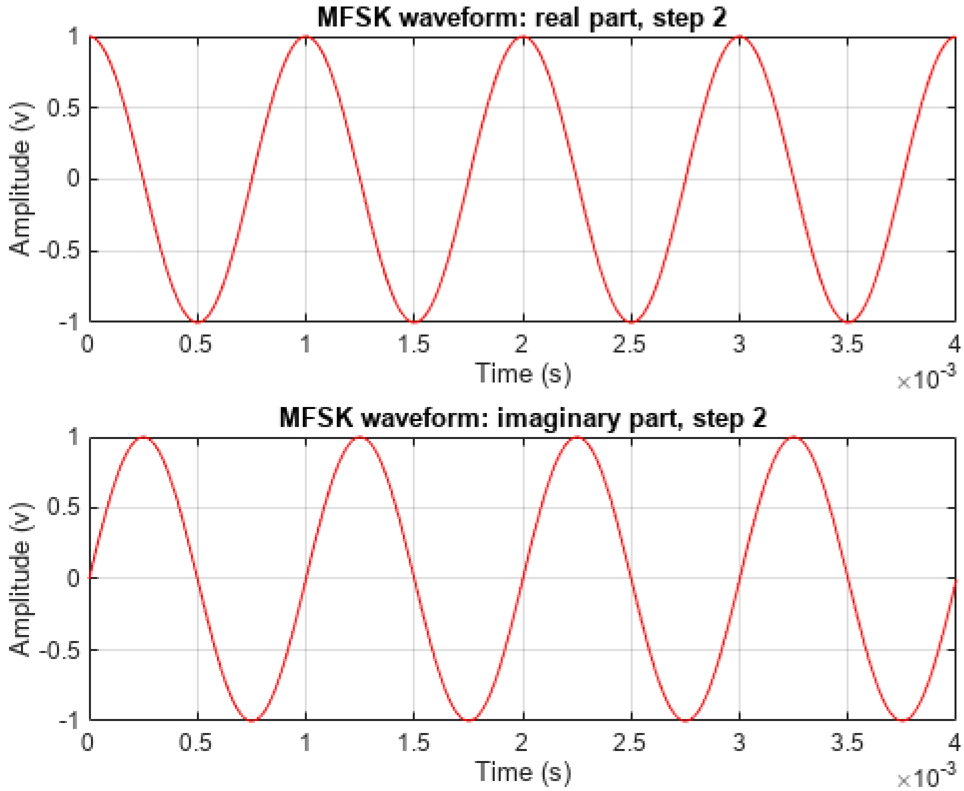
Plot MFSK Waveform

Construct an MFSK waveform with a sample rate of 1 MHz and a sweep bandwidth of 0.1 MHz. Assume 52 steps with a step time of 4 milliseconds. Set the frequency offset to 1 kHz. There are 4000 samples per step.

```
fs = 1e6;
fsweep = 1e5;
tstep = 4e-3;
numsteps = 52;
foffset = 1000;
noutputsteps = 4;
sMFSK = phased.MFSKWaveform('SampleRate',fs,...
    'SweepBandwidth',fsweep,...
    'StepTime',tstep,...
    'StepsPerSweep',numsteps,...
    'FrequencyOffset',foffset,...
    'OutputFormat','Steps',...
    'NumSteps',noutputsteps);
```

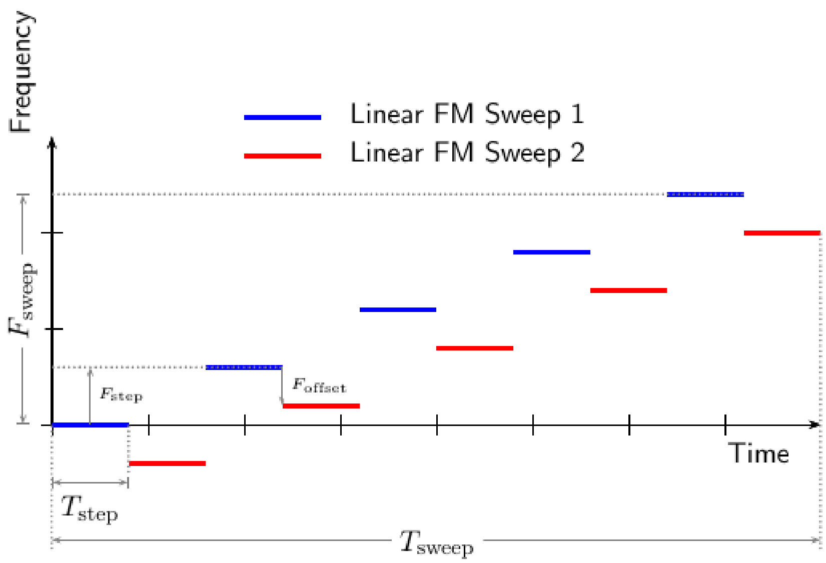
Plot the real and imaginary components of the second step of the waveform using the `plot` method. Set the plot color to red.

```
plot(sMFSK,'PlotType','complex','StepIdx',2,'r')
```



Algorithms

An MFSK waveform consists of two interleaved stepped-frequency sequences, as shown in this time-frequency diagram.



Each sequence is a set of continuous waveform (CW) signals increasing in frequency. The offset, F_{offset}, between the two sequences is constant and can be positive or negative. A complete waveform consists of an even number of steps, N, of equal duration, T_{step}. Then, each sequence consists of $N/2$ steps. The sweep frequency, F_{sweep}, is the difference between the lowest and highest frequency of either sequence. F_{sweep} is always positive, indicating increasing frequency. The frequency difference between successive steps of each sequence is given by

$F_{step} = F_{sweep}/(N/2-1)$.

The lowest frequency of the first sequence is always 0 hertz and corresponds to the carrier frequency of the bandpass signal. The lowest frequency of the second sequence can be positive or negative and is equal to F_{offset}. Negative frequencies correspond to bandpass frequencies that are lower than the carrier frequency. The duration of the waveform is given by $T_{sweep} = N * T_{step}$. The System object properties corresponding to the signal parameters are

Signal Parameter	Property
F_{sweep}	`'SweepBandwidth'`
T_{step}	`'StepTime'`
N	`'StepsPerSweep'`
F_{offset}	`'FrequencyOffset'`

Version History

Introduced in R2015a

References

[1] Meinecke, Marc-Michale, and Hermann Rohling, “Combination of LFMCW and FSK Modulation Principles for Automotive Radar Systems.” *German Radar Symposium GRS2000.* 2000.

[2] Rohling, Hermann, and Marc-Michale Meinecke. “Waveform Design Principles for Automotive Radar Systems”. *CIE International Conference on Radar*. 2001.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `plot` method is not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.LinearFMWaveform` | `phased.PhaseCodedWaveform` | `phased.RectangularWaveform` | `phased.SteppedFMWaveform` | `phased.FMCWWaveform` | `phased.MatchedFilter`

Topics
“Simultaneous Range and Speed Estimation Using MFSK Waveform”

plot

System object: `phased.MFSKWaveform`
Package: `phased`

Plot continuous MFSK waveform

Syntax

```
plot(sMFSK)
plot(sMFSK,Name,Value)
plot(sMFSK,Name,Value,LineSpec)
h = plot(___)
```

Description

`plot(sMFSK)` plots the real part of the waveform specified by `sMFSK`.

`plot(sMFSK,Name,Value)` plots the waveform with additional options specified by one or more `Name,Value` pair arguments.

`plot(sMFSK,Name,Value,LineSpec)` specifies the same line color, line style, or marker options that are available in the MATLAB `plot` function.

`h = plot(___)` returns the line handle in the figure.

Input Arguments

`sMFSK` — MFSK waveform
MFSK waveform System object

MFSK waveform, specified as a `phased.MFSKWaveform` System object.

Example: `sMFSK = phased.MFSKWaveform;`

`LineSpec` — Plot style
`'b'` (default) | character vector

Plot style, specified as a character vector. You can specify the same line color, style, or marker options that are available in the MATLAB `plot` function. If you specify a `PlotType` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Example: `'k.'`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

PlotType — Waveform component to plot
'real' (default) | 'imag' | 'complex'

Waveform component to plot, specified as the comma-separated pair consisting of 'PlotType' and one of the following:

- 'real' — Plots the real part of the waveform
- 'imag' — Plots the imaginary part of the waveform
- 'complex' — Plots both parts of the waveform

Example: 'PlotType','complex'

StepIdx — Index of step
1 (default) | positive integer

Index of the step to plot, specified as the comma-separated pair consisting of 'StepIdx' and a positive integer. If you specify a 'StepIdx' value greater than 'StepsPerSweep', the frequency corresponds to the mod('StepIdx','StepsPerSweep') value.

Output Arguments

h — Plot handle
double

Plot handle(s) to the line or lines in the figure, returned as a double. When PlotType is set to 'complex', h is a 2-by-1 column vector. The first and second elements of this vector are the handles to the lines in the real and imaginary subplots, respectively.

Examples

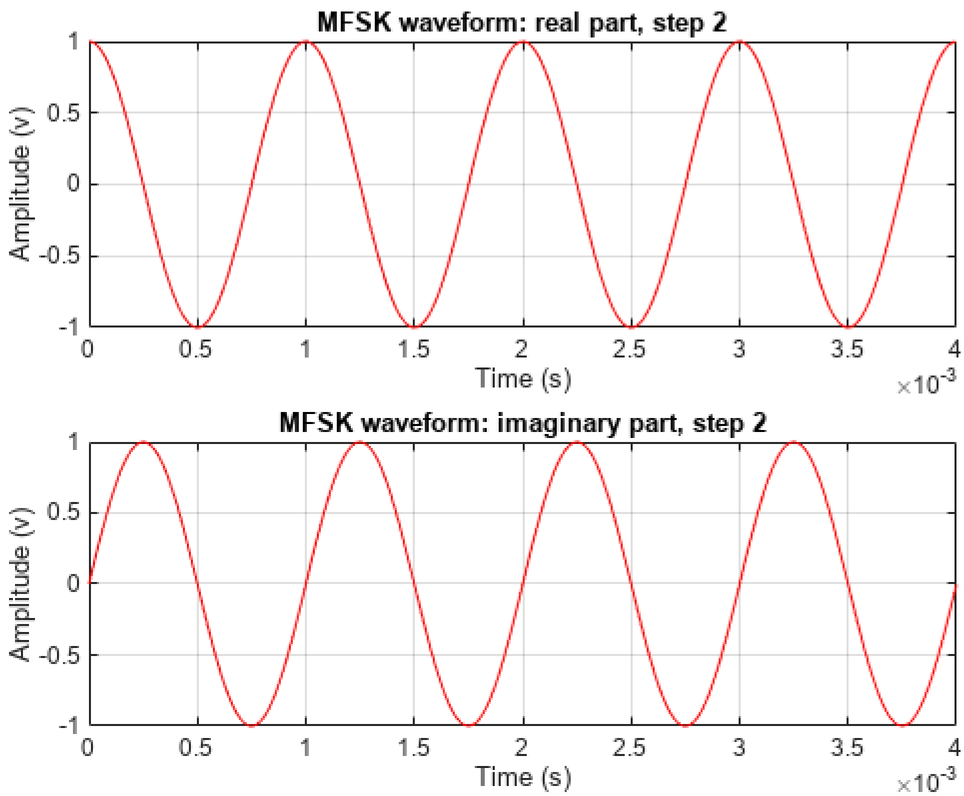
Plot MFSK Waveform

Construct an MFSK waveform with a sample rate of 1 MHz and a sweep bandwidth of 0.1 MHz. Assume 52 steps with a step time of 4 milliseconds. Set the frequency offset to 1 kHz. There are 4000 samples per step.

```
fs = 1e6;
fsweep = 1e5;
tstep = 4e-3;
numsteps = 52;
foffset = 1000;
noutputsteps = 4;
sMFSK = phased.MFSKWaveform('SampleRate',fs,...
    'SweepBandwidth',fsweep,...
    'StepTime',tstep,...
    'StepsPerSweep',numsteps,...
    'FrequencyOffset',foffset,...
    'OutputFormat','Steps',...
    'NumSteps',noutputsteps);
```

Plot the real and imaginary components of the second step of the waveform using the plot method. Set the plot color to red.

```
plot(sMFSK,'PlotType','complex','StepIdx',2,'r')
```



Version History

Introduced in R2015a

reset

System object: `phased.MFSKWaveform`
Package: `phased`

Reset states of the MFSK waveform object

Syntax

```
reset(sMFSK)
```

Description

`reset(sMFSK)` resets the internal states of the `phased.MFSKWaveform` object, `sMFSK`, to their initial values.

Input Arguments

sMFSK — MFSK waveform
System object

MFSK waveform, specified as a `phased.MFSKWaveform` System object.

Example: `sMFSK= phased.MFSKWaveform;`

Version History

Introduced in R2015a

step

System object: `phased.MFSKWaveform`
Package: `phased`

Samples of continuous MFSK waveform

Syntax

`Y = step(sMFSK)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

`Y = step(sMFSK)` returns samples of the MFSK waveform in a *N*-by-1 complex valued column vector, `Y`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`sMFSK` — MFSK waveform

System object

MFSK waveform, specified as a `phased.MFSKWaveform` System object.

Example: `sMFSK= phased.MFSKWaveform;`

Output Arguments

`Y` — Output samples

N-by-1 complex valued vector

Output samples of MFSK waveform, returned as an *N*-by-1 complex valued vector. When the `step` method reaches the end of the waveform, the output samples wrap around from the start of the waveform, yielding a periodic waveform.

Examples

Construct MFSK Step Output

Construct an MFSK waveform with a sample rate of 1 MHz and a sweep bandwidth of 0.1 MHz. Assume 52 steps, with a step time of 4 milliseconds. Set the frequency offset to 1 kHz. There are 4000 samples per step.

```
fs = 1e6;
fsweep = 1e5;
tstep = 40e-4;
numsteps = 52;
foffset = 1000;
noutputsteps = 4;
sMFSK = phased.MFSKWaveform('SampleRate',fs,...
    'SweepBandwidth',fsweep,...
    'StepTime',tstep,...
    'StepsPerSweep',numsteps,...
    'FrequencyOffset',foffset,...
    'OutputFormat','Steps',...
    'NumSteps',noutputsteps);
```

Call the step method to retrieve the samples for the four steps.

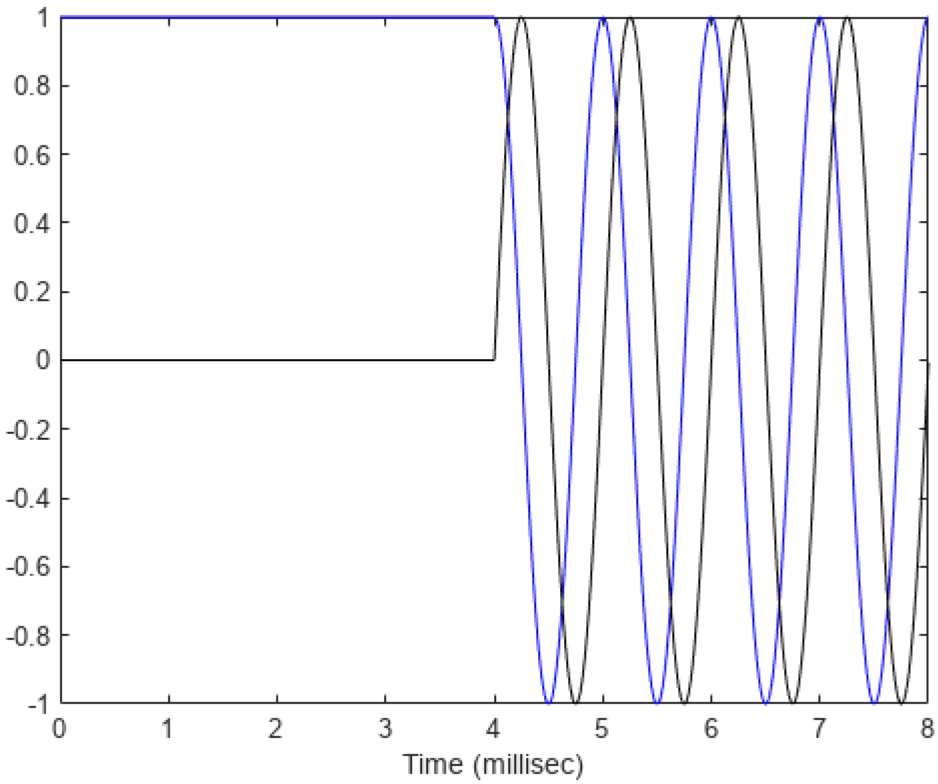
```
z = step(sMFSK);
```

Plot the real and imaginary parts of the first two steps.

```
samplesperstep = fs*tstep;
disp(samplesperstep)
```

```
        4000
```

```
idx = [1:2*samplesperstep]';
time = idx/fs*1000;
plot(time,real(z(idx)),'b',time,imag(z(idx)),'k');
xlabel('Time (millisec)')
```

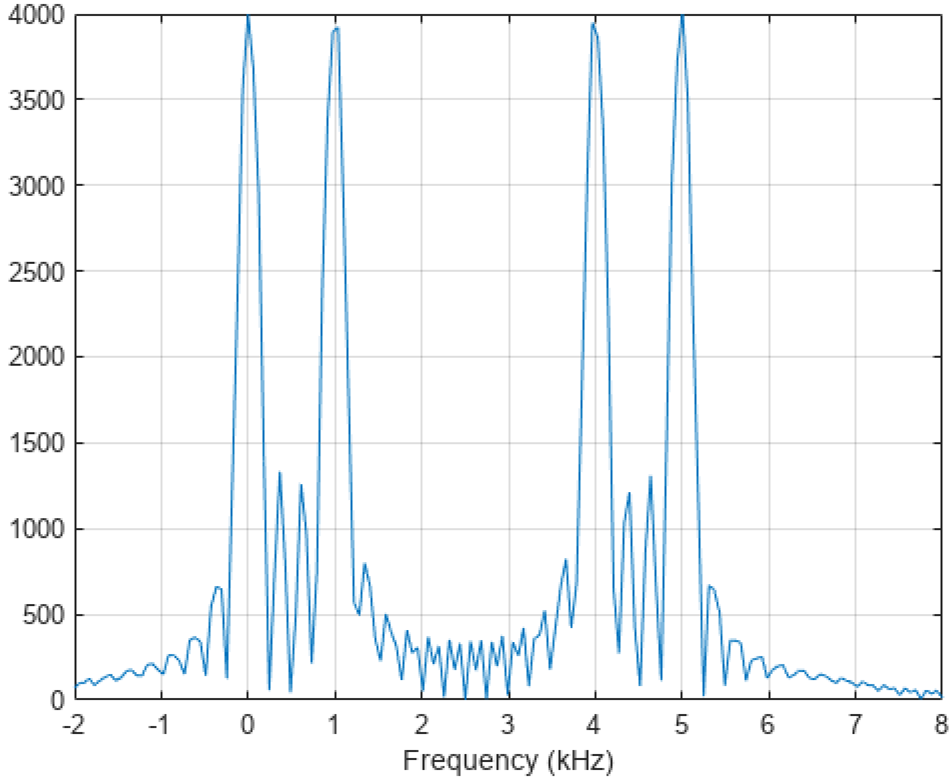


Compute the FFT of all the data.

```
n = size(z,1);
nfft = 2^ceil(log2(n));
Y = fftshift(fft(z,nfft));
```

Plot the magnitudes of the spectrum.

```
fmax = fs/2;
ft = [-nfft/2:nfft/2-1]*fmax/(nfft/2);
figure(2);
hp = plot(ft/1000,abs(Y));
axis([-2,8,-1,4000]);
xlabel('Frequency (kHz)')
grid
```



The plot shows two pairs of peaks. The first pair lies at 0 Hz and 1000 Hz. The second pair lies at 4000 Hz and 5000 Hz. The frequency offset is 1000 Hz.

Compute the frequency increase to the second pair off peaks.

```
fdelta = fsweep/(numsteps/2-1);
disp(fdelta)
```

```
        4000
```

The increase agrees with the location of the second pair of peaks in the FFT spectrum.

MFSK Samples per Sweep

Construct an MFSK waveform with a sample rate of 1 MHz and a sweep bandwidth of 0.1 MHz. Assume 52 steps with a step time of 400 microseconds. Set the frequency offset to 1 kHz. Find the number of samples returned when the `OutputFormat` property is set to return the samples for one sweep.

```
fs = 1e6;
fsweep = 1e5;
tstep = 40e-4;
numsteps = 52;
foffset = 1000;
noutputsweeps = 1;
```

```
sMFSK = phased.MFSKWaveform('SampleRate',fs,...
    'SweepBandwidth',fsweep,...
    'StepTime',tstep,...
    'StepsPerSweep',numsteps,...
    'FrequencyOffset',foffset,...
    'OutputFormat','Sweeps',...
    'NumSweeps',noutputsweeps);
```

Call the `step` method to retrieve the samples for the four steps.

```
z = step(sMFSK);
```

Count the number of samples in a sweep.

```
samplespersweep = fs*tstep*numsteps;
disp(samplespersweep)
```

```
      208000
```

Verify that this value agrees with the number of samples returned by the `step` method.

```
disp(size(z))
```

```
      208000           1
```

Version History

Introduced in R2015a

phased.MVDRBeamformer

Package: `phased`

Narrowband minimum-variance distortionless-response beamformer

Description

The phased.MVDRBeamformer System object implements a narrowband minimum-variance distortionless-response (MVDR) beamformer. The MVDR beamformer is also called the Capon beamformer. An MVDR beamformer belongs to the family of constrained optimization beamformers.

To beamform signals arriving at an array:

1. Create the `phased.MVDRBeamformer` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
beamformer = phased.MVDRBeamformer
beamformer = phased.MVDRBeamformer(Name,Value)
```

Description

`beamformer = phased.MVDRBeamformer` creates an MVDR beamformer System object, `beamformer`, with default property values.

`beamformer = phased.MVDRBeamformer(Name,Value)` creates an MVDR beamformer with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `beamformer = phased.MVDRBeamformer('SensorArray',phased.URA,'OperatingFrequency',300e6)` sets the sensor array to a uniform rectangular array (URA) with default URA property values. The beamformer has an operating frequency of 300 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

SensorArray — Sensor array
phased.ULA array with default property values (default) | Phased Array System Toolbox array

Sensor array, specified as an array System object belonging to Phased Array System Toolbox. The sensor array can contain subarrays.

Example: phased.URA

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed').

Example: 3e8

Data Types: single | double

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: 1e9

Data Types: single | double

DiagonalLoadingFactor — Diagonal loading factor
0 (default) | nonnegative scalar

Diagonal loading factor, specified as a nonnegative scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample size is small. A small sample size can lead to an inaccurate estimate of the covariance matrix. Diagonal loading also provides robustness due to steering vector errors. The diagonal loading technique adds a positive scalar multiple of the identity matrix to the sample covariance matrix.

Tunable: Yes

Data Types: single | double

TrainingInputPort — Enable training data input
false (default) | true

Enable training data input, specified as false or true. When you set this property to true, use the training data input argument, XT, when running the object. Set this property to false to use the input data, X, as the training data.

Data Types: logical

DirectionSource — Source of beamforming direction
'Property' (default) | 'Input port'

Source of beamforming direction, specified as 'Property' or 'Input port'. Specify whether the beamforming direction comes from the Direction property of this object or from the input argument, ANG. Values of this property are:

'Property'	Specify the beamforming direction using the Direction property.
'Input port'	Specify the beamforming direction using the input argument, ANG.

Data Types: char

Direction — Beamforming directions
[0;0] (default) | real-valued 2-by-1 vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 vector or a real-valued 2-by-*L* matrix. For a matrix, each column specifies a different beamforming direction. Each column has the form [AzimuthAngle;ElevationAngle]. Azimuth angles must lie between –180° and 180° and elevation angles must lie between –90° and 90°. All angles are defined with respect to the local coordinate system of the array. Units are in degrees.

Example: [40;30]

Dependencies

To enable this property, set the DirectionSource property to 'Property'.

Data Types: single | double

NumPhaseShifterBits — Number of phase shifter quantization bits
0 (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights, specified as a nonnegative integer. A value of zero indicates that no quantization is performed.

Example: 5

Data Types: single | double

WeightsOutputPort — Enable beamforming weights output
false (default) | true

Enable the output of beamforming weights, specified as false or true. To obtain the beamforming weights, set this property to true and use the corresponding output argument, W. If you do not want to obtain the weights, set this property to false.

Data Types: logical

Usage

Syntax

```
Y = beamformer(X)
Y = beamformer(X,XT)
Y = beamformer(X,ANG)
Y = beamformer(X,XT,ANG)
[Y,W] = beamformer( ___ )
```

Description

`Y = beamformer(X)` performs MVDR beamforming on the input signal, `X`, and returns the beamformed output in `Y`. This syntax uses `X` as training samples to calculate the beamforming weights.

`Y = beamformer(X,XT)` uses `XT` as training samples to calculate the beamforming weights. To use this syntax, set the TrainingInputPort property to `true`.

`Y = beamformer(X,ANG)` uses `ANG` as the beamforming direction. To use this syntax, set the DirectionSource property to `'Input port'`.

`Y = beamformer(X,XT,ANG)` combines all input arguments. To use this syntax, set the `TrainingInputPort` property to `true` and set the DirectionSource property to `'Input port'`.

`[Y,W] = beamformer( ___ )` returns the beamforming weights, `W`. To use this syntax, set the WeightsOutputPort property to `true`.

Input Arguments

X — Input signal
complex-valued *M*-by-*N* matrix

Input signal, specified as a complex-valued *M*-by-*N* matrix. *N* is the number of array elements. If the sensor array contains subarrays, *N* is the number of subarrays. If you set TrainingInputPort to `false`, *M* must be larger than *N*; otherwise, *M* can be any positive integer.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`
Complex Number Support: Yes

XT — Training data
complex-valued *P*-by-*N* matrix

Training data, specified as a complex-valued *P*-by-*N* matrix. If the sensor array contains subarrays, *N* is the number of subarrays; otherwise, *N* is the number of elements. *P* must be larger than *N*.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1 0.5 2.6; 2 -0.2 0; 3 -2 -1]`

Dependencies

To enable this argument, set the TrainingInputPort property to `true`.

Data Types: `single` | `double`
Complex Number Support: Yes

ANG — Beamforming directions
`[0;0]` (default) | real-valued 2-by-1 column vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 column vector, or 2-by-*L* matrix. *L* is the number of beamforming directions. Each column has the form `[AzimuthAngle;ElevationAngle]`.

Units are in degrees. Each azimuth angle must lie between –180° and 180°, and each elevation angle must lie between –90° and 90°.

Example: `[40;10]`

Dependencies

To enable this argument, set the DirectionSource property to `'Input port'`.

Data Types: `single` | `double`

Output Arguments

Y — Beamformed output
complex-valued *M*-by-*L* matrix

Beamformed output, returned as a complex-valued *M*-by-*L* matrix, where *M* is the number of rows of `X` and *L* is the number of beamforming directions.

Data Types: `single` | `double`
Complex Number Support: Yes

`W` — Beamforming weights
complex-valued *N*-by-*L* matrix.

Beamforming weights, returned as a complex-valued *N*-by-*L* matrix. If the sensor array contains subarrays, *N* is the number of subarrays; otherwise, *N* is the number of elements. *L* is the number of beamforming directions.

Dependencies

To enable this output, set the WeightsOutputPort property to `true`.

Data Types: `single` | `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

MVDR Beamforming

Apply an MVDR beamformer to a 5-element ULA. The incident angle of the signal is 45 degrees in azimuth and 0 degree in elevation. The signal frequency is .01 hertz. The carrier frequency is 300 MHz.

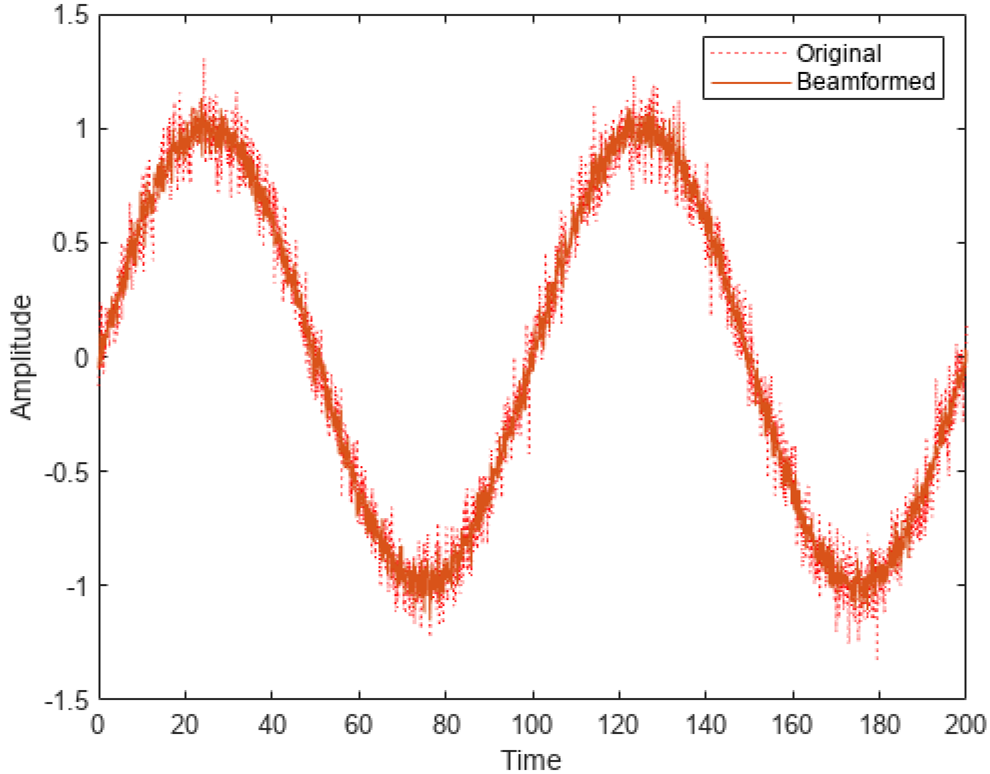
```
t = [0:.1:200]';
fr = .01;
xm = sin(2*pi*fr*t);
c = physconst('LightSpeed');
fc = 300e6;
rng('default');
incidentAngle = [45;0];
array = phased.ULA('NumElements',5,'ElementSpacing',0.5);
x = collectPlaneWave(array,xm,incidentAngle,fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;
```

Compute the beamforming weights.

```
beamformer = phased.MVDRBeamformer('SensorArray',array,...
    'PropagationSpeed',c,'OperatingFrequency',fc,...
    'Direction',incidentAngle,'WeightsOutputPort',true);
[y,w] = beamformer(rx);
```

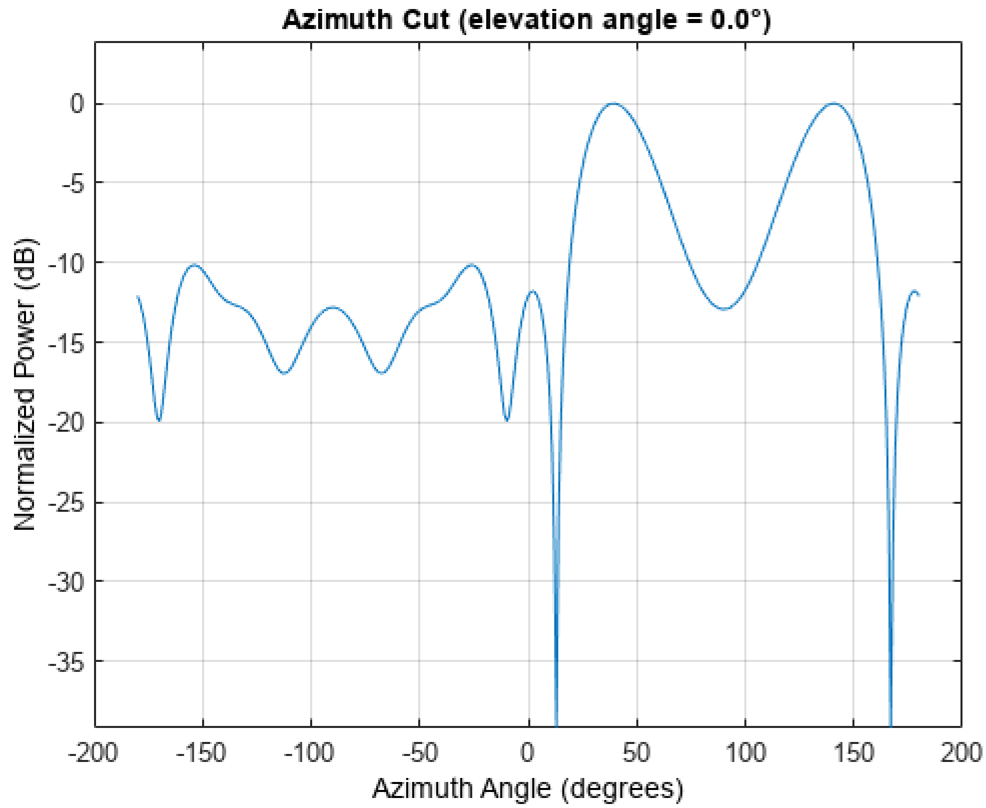
Plot the signals.

```
plot(t,real(rx(:,3)),'r:',t,real(y))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Plot the array response pattern using the MVDR weights.

```
pattern(array,fc,[-180:180],0,'PropagationSpeed',c,...
    'Weights',w,'CoordinateSystem','rectangular',...
    'Type','powerdb');
```



Algorithms

MVDR Beamforming

The MVDR beamformer maximizes the signal to noise ratio.

Start with a signal arriving at the elements of an array. Assume that X is a complex-valued N-by-M data matrix representing the arrival of signals at an array. N is the number of sensors in the array and M is the number of samples or snapshots per signal. For mathematical convenience, this matrix is the transpose of the matrix specified in the X argument. Each row of X represents a time series of data for the corresponding array. The signal-to-noise ratio of a signal is given here.

$$SNR = \frac{\left|w^H s\right|^2}{w^H R_{I+N} w}$$

Properly, the covariance matrix in the denominator is the covariance matrix for the noise and any interferers. You can vary the scale of w without affecting the SNR. Therefore, you can choose the normalization for w so that The MVDR estimator weights for beamforming are $w = R^{-1}v/v^H R v$ where R is the data covariance matrix $R = E[xx^H]$.

Diagonal Loading

Diagonal loading provides beamformer robustness due to small sample size and steering vector errors.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

[2] Frost, O. "An Algorithm For Linearly Constrained Adaptive Array Processing", *Proceedings of the IEEE*. Vol. 60, Number 8, August, 1972, pp. 926–935.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.FrostBeamformer` | `phased.PhaseShiftBeamformer` | `phased.LCMVBeamformer` | `phased.SubbandMVDRBeamformer`

phased.MVDREstimator

Package: `phased`

MVDR (Capon) spatial spectrum estimator for ULA

Description

The `MVDREstimator` object computes a minimum variance distortionless response (MVDR) spatial spectrum estimate for a uniform linear array. This DOA estimator is also referred to as a Capon DOA estimator.

To estimate the spatial spectrum:

1. Define and set up your MVDR spatial spectrum estimator. See "Construction" on page 1-1002.
2. Call `step` to estimate the spatial spectrum according to the properties of `phased.MVDREstimator`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.MVDREstimator` creates an MVDR spatial spectrum estimator System object, `H`. The object estimates the incoming signal's spatial spectrum using a narrowband MVDR beamformer for a uniform linear array (ULA).

`H = phased.MVDREstimator(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be a `phased.ULA` object.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz.

Default: `3e8`

NumPhaseShifterBits

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Default: `0`

ForwardBackwardAveraging

Perform forward-backward averaging

Set this property to `true` to use forward-backward averaging to estimate the covariance matrix for sensor arrays with conjugate symmetric array manifold.

Default: `false`

SpatialSmoothing

Spatial smoothing

Specify the number of averaging used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each additional smoothing handles one extra coherent source, but reduces the effective number of element by 1. The maximum value of this property is M–2, where M is the number of sensors.

Default: `0`, indicating no spatial smoothing

ScanAngles

Scan angles

Specify the scan angles (in degrees) as a real vector. The angles are broadside angles and must be between –90 and 90, inclusive. You must specify the angles in ascending order.

Default: `-90:90`

DOAOutputPort

Enable DOA output

To obtain the signal's direction of arrival (DOA), set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the DOA, set this property to `false`.

Default: `false`

`NumSignals`

Number of signals

Specify the number of signals for DOA estimation as a positive scalar integer. This property applies when you set the `DOAOutputPort` property to true.

Default: 1

Methods

plotSpectrum	Plot spatial spectrum
reset	Reset states of MVDR spatial spectrum estimator object
step	Perform spatial spectrum estimation

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Estimate DOA of Two Signals Using MVDR

First, estimate the DOAs of two signals received by a standard 10-element ULA with element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 60° in azimuth and −5° in elevation. Then, plot the MVDR spatial spectrum.

Create the signals with added noise. Then, create the ULA System object™.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
array.Element.FrequencyRange = [100e6 300e6];
fc = 150.0e6;
x = collectPlaneWave(array,[x1 x2],[10 20;60 -5]',fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
```

Construct MVDR estimator System object.

```
estimator = phased.MVDREstimator('SensorArray',array,...
    'OperatingFrequency',fc,'DOAOutputPort',true,'NumSignals',2);
```

Estimate the DOAs.

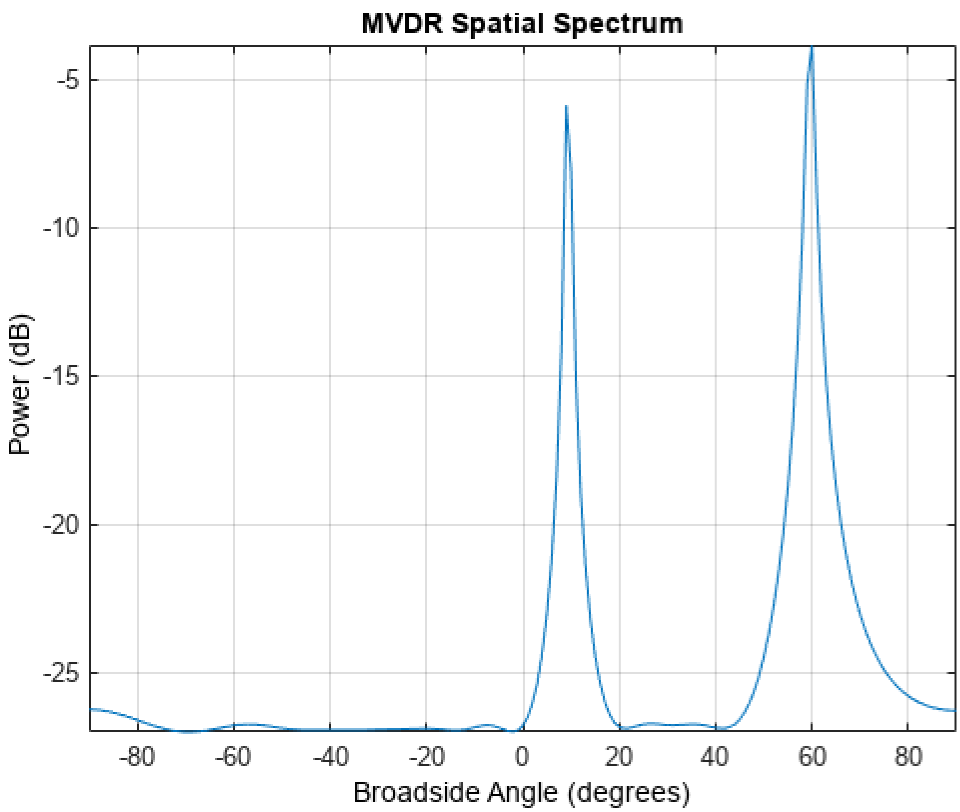
```
[y,doas] = estimator(x + noise);
doas = broadside2az(sort(doas),[20 -5])
```

```
doas = 1×2
```

```
    9.5829   60.3813
```

Plot the spectrum.

```
plotSpectrum(estimator)
```



Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`broadside2az` | `phased.MVDREstimator2D`

plotSpectrum

System object: `phased.MVDREstimator`
Package: `phased`

Plot spatial spectrum

Syntax

```
plotSpectrum(estimator)
plotSpectrum(estimator,Name,Value)
hl = plotSpectrum( ___ )
```

Description

`plotSpectrum(estimator)` plots the spatial spectrum resulting from the most recent execution of the object.

`plotSpectrum(estimator,Name,Value)` plots the spatial spectrum with additional options specified by one or more `Name,Value` pair arguments.

`hl = plotSpectrum( ___ )` returns the line handle in the figure.

Input Arguments

`H`

Spatial spectrum estimator object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`NormalizeResponse`

Set this value to `true` to plot the normalized spectrum. Setting this value to `false` plots the spectrum without normalization.

Default: `false`

`Title`

Character vector to use as figure title.

Default: `''`

Unit

Plot units, specified as `'db'`, `'mag'`, or `'pow'`.

Default: `'db'`

Examples

Estimate DOA of Two Signals Using MVDR

First, estimate the DOAs of two signals received by a standard 10-element ULA with element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 60° in azimuth and −5° in elevation. Then, plot the MVDR spatial spectrum.

Create the signals with added noise. Then, create the ULA System object™.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
array.Element.FrequencyRange = [100e6 300e6];
fc = 150.0e6;
x = collectPlaneWave(array,[x1 x2],[10 20;60 -5]',fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
```

Construct MVDR estimator System object.

```
estimator = phased.MVDREstimator('SensorArray',array,...
    'OperatingFrequency',fc,'DOAOutputPort',true,'NumSignals',2);
```

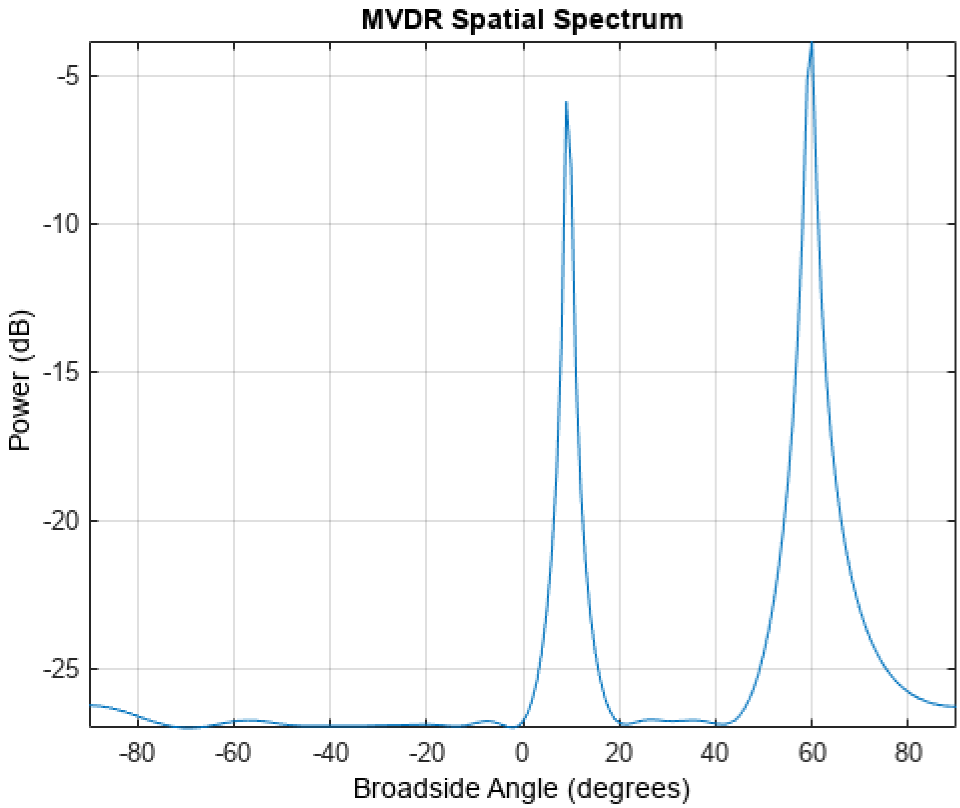
Estimate the DOAs.

```
[y,doas] = estimator(x + noise);
doas = broadside2az(sort(doas),[20 -5])
```

```
doas = 1×2

    9.5829   60.3813
```

Plot the spectrum.

```
plotSpectrum(estimator)
```


MVDR Spatial Spectrum
Power (dB)
Broadside Angle (degrees)
-5
-10
-15
-20
-25
-80
-60
-40
-20
0
20
40
60
80

reset

System object: `phased.MVDREstimator`
Package: `phased`

Reset states of MVDR spatial spectrum estimator object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `MVDREstimator` object, `H`.

step

System object: `phased.MVDREstimator`
Package: `phased`

Perform spatial spectrum estimation

Syntax

```
Y = step(H,X)
[Y,ANG] = step(H,X)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` estimates the spatial spectrum from `X` using the estimator `H`. `X` is a matrix whose columns correspond to channels. `Y` is a column vector representing the magnitude of the estimated spatial spectrum.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`[Y,ANG] = step(H,X)` returns additional output `ANG` as the signal's direction of arrival (DOA) when the `DOAOutputPort` property is true. `ANG` is a row vector of the estimated broadside angles (in degrees).

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Estimate DOA of Two Signals Using MVDR

First, estimate the DOAs of two signals received by a standard 10-element ULA with element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 60° in azimuth and −5° in elevation. Then, plot the MVDR spatial spectrum.

Create the signals with added noise. Then, create the ULA System object™.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
array.Element.FrequencyRange = [100e6 300e6];
fc = 150.0e6;
x = collectPlaneWave(array,[x1 x2],[10 20;60 -5]',fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
```

Construct MVDR estimator System object.

```
estimator = phased.MVDREstimator('SensorArray',array,...
    'OperatingFrequency',fc,'DOAOutputPort',true,'NumSignals',2);
```

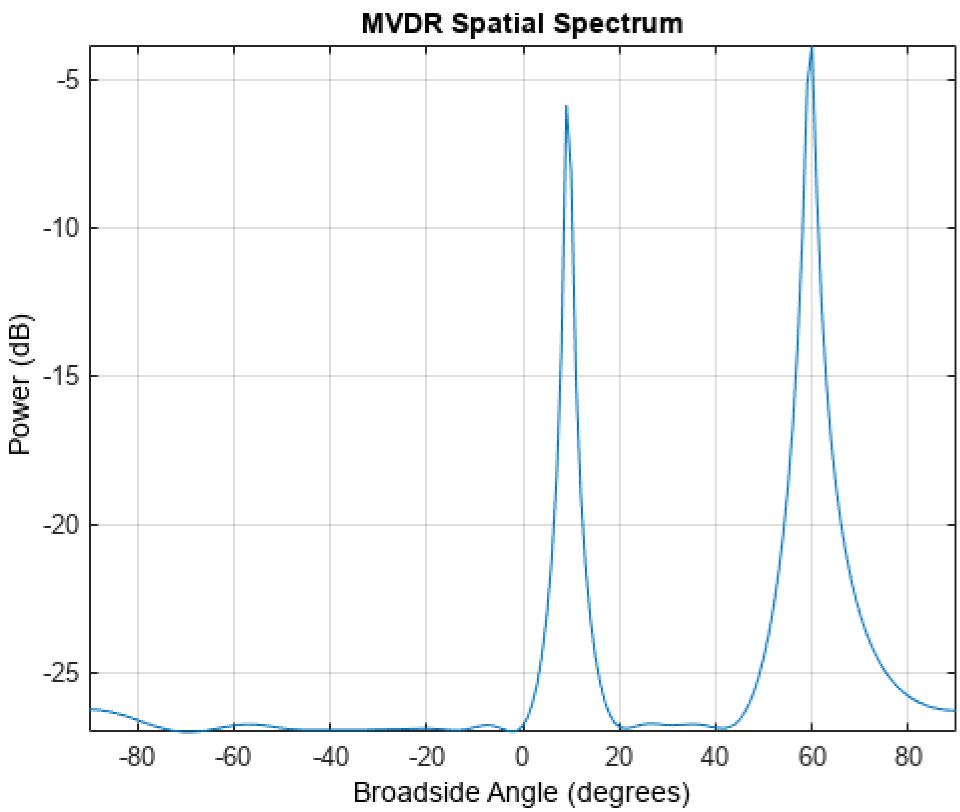
Estimate the DOAs.

```
[y,doas] = estimator(x + noise);
doas = broadside2az(sort(doas),[20 -5])
```

```
doas = 1×2

    9.5829   60.3813
```

Plot the spectrum.

```
plotSpectrum(estimator)
```



phased.MVDREstimator2D

Package: `phased`

2-D MVDR (Capon) spatial spectrum estimator

Description

The `MVDREstimator2D` object computes a 2-D minimum variance distortionless response (MVDR) spatial spectrum estimate. This DOA estimator is also referred to as a Capon estimator.

To estimate the spatial spectrum:

1. Define and set up your 2-D MVDR spatial spectrum estimator. See "Construction" on page 1-1014.
2. Call `step` to estimate the spatial spectrum according to the properties of `phased.MVDREstimator2D`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.MVDREstimator2D` creates a 2-D MVDR spatial spectrum estimator System object, `H`. The object estimates the signal's spatial spectrum using a narrowband MVDR beamformer.

`H = phased.MVDREstimator2D(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be an array object in the `phased` package. The array cannot contain subarrays.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

NumPhaseShifterBits

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed. You can specify this property as single or double precision.

Default: `0`

ForwardBackwardAveraging

Perform forward-backward averaging

Set this property to `true` to use forward-backward averaging to estimate the covariance matrix for sensor arrays with conjugate symmetric array manifold.

Default: `false`

AzimuthScanAngles

Azimuth scan angles (degrees)

Specify the azimuth scan angles (in degrees) as a real vector. The angles must be between –180 and 180, inclusive. You must specify the angles in ascending order. You can specify this property as single or double precision.

Default: `-90:90`

ElevationScanAngles

Elevation scan angles

Specify the elevation scan angles (in degrees) as a real vector or scalar. The angles must be between –90 and 90, inclusive. You must specify the angles in ascending order. You can specify this property as single or double precision.

Default: `0`

DOAOutputPort

Enable DOA output

To obtain the signal's direction of arrival (DOA), set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the DOA, set this property to `false`.

Default: `false`

`NumSignals`

Number of signals

Specify the number of signals for DOA estimation as a positive scalar integer. This property applies when you set the `DOAOutputPort` property to `true`. You can specify this property as single or double precision.

Default: 1

Methods

plotSpectrum	Plot spatial spectrum
reset	Reset states of 2-D MVDR spatial spectrum estimator object
step	Perform spatial spectrum estimation

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Estimate DOA of Two Signals Arriving at URA

Estimate the DOAs of two signals received by a 50-element URA with a rectangular lattice. The antenna operating frequency is 150 MHz. The actual direction of the first signal is −37° in azimuth and 0° in elevation. The direction of the second signal is 17° in azimuth and 20° degrees in elevation. Then, plot the spatial spectrum.

Create the arriving signals.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
array.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(array,[x1 x2],[-37 0;17 20]',fc);
```

Add noise.

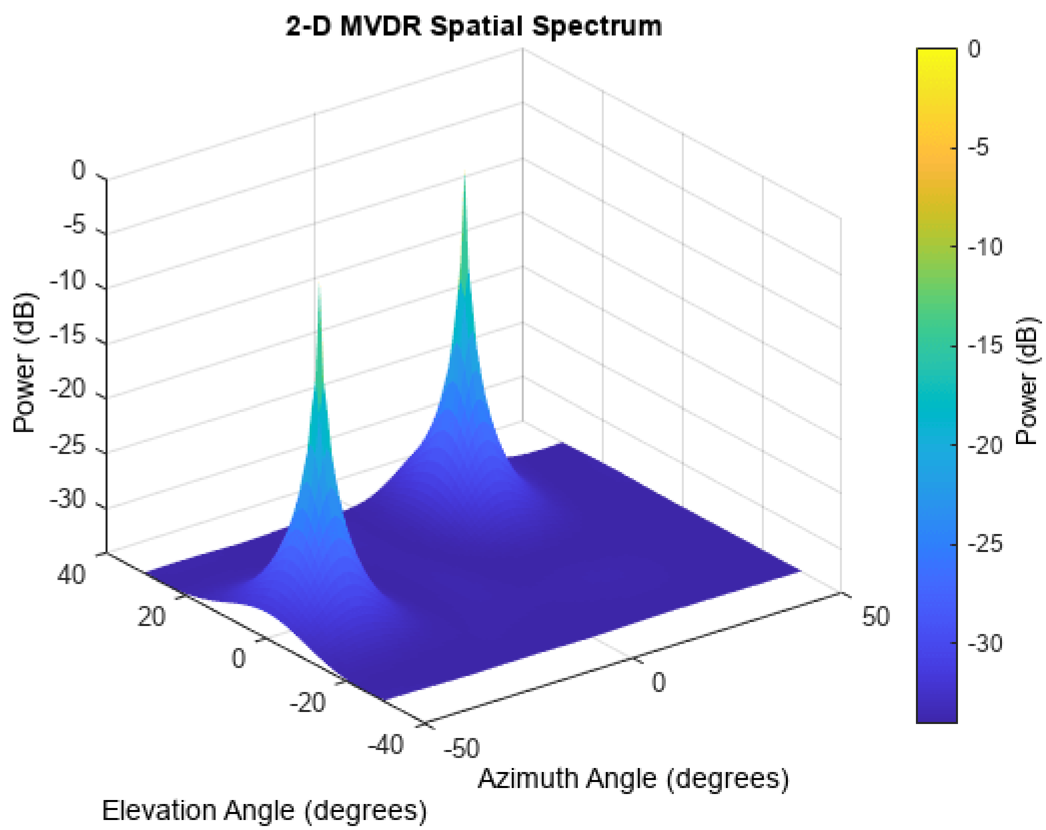
```
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
```

Create the MVDR DOA estimator and estimate the DOAs.

```
estimator = phased.MVDREstimator2D('SensorArray',array,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
[~,doas] = estimator(x + noise);
```

Plot the spectrum.

```
plotSpectrum(estimator)
```



Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

phased.MVDREstimator | uv2azel | phitheta2azel

plotSpectrum

System object: `phased.MVDREstimator2D`
Package: `phased`

Plot spatial spectrum

Syntax

```
plotSpectrum(estimator)
plotSpectrum(estimator,Name,Value)
hl = plotSpectrum( ___ )
```

Description

`plotSpectrum(estimator)` plots the spatial spectrum resulting from the most recent execution of the object.

`plotSpectrum(estimator,Name,Value)` plots the spatial spectrum with additional options specified by one or more `Name,Value` pair arguments.

`hl = plotSpectrum( ___ )` returns the line handle in the figure.

Input Arguments

`H`

Spatial spectrum estimator object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`NormalizeResponse`

Set this value to `true` to plot the normalized spectrum. Setting this value to `false` plots the spectrum without normalization.

Default: `false`

`Title`

Character vector to use as figure title.

Default: `''`

Unit

Plot units, specified as `'db'`, `'mag'`, or `'pow'`.

Default: `'db'`

Examples

Estimate DOA Using 2D MVDR

Estimate the DOAs of two signals received by a 50-element URA with a rectangular lattice. The antenna operating frequency is 150 MHz. The actual direction of the first signal is -37° in azimuth and 0° in elevation. The direction of the second signal is 17° in azimuth and 20° in elevation.

Create signals sampled at 8 kHz.

```
fc = 150e6;
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
array.Element.FrequencyRange = [100e6 300e6];
x = collectPlaneWave(array,[x1 x2],[-37 0;17 20]',fc);
```

Add complex noise.

```
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
```

Create the MVDR DOA estimator for URA.

```
estimator = phased.MVDREstimator2D('SensorArray',array,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
```

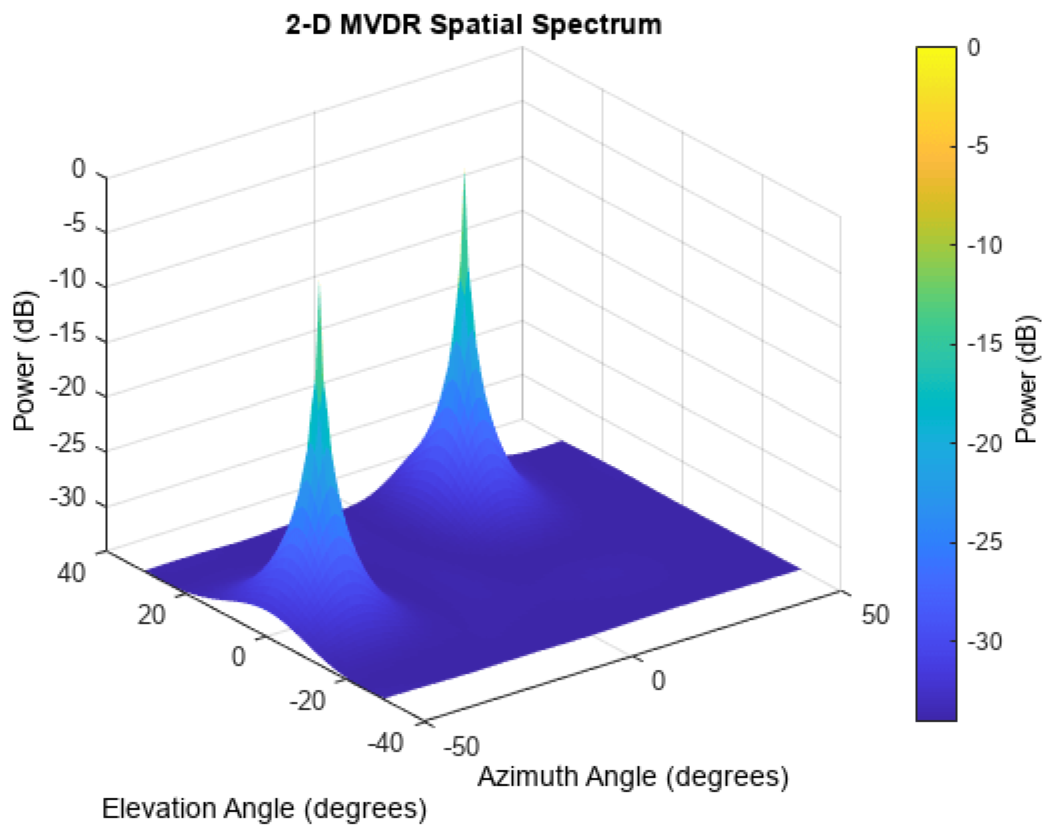
Use the `step` method to the DOA estimates.

```
[~,doas] = estimator(x + noise)
```

```
doas = 2×2

    17   -37
    20     0
```

Plot the spectrum.

```
plotSpectrum(estimator)
```



2-D MVDR Spatial Spectrum
Power (dB)
0
-5
-10
-15
-20
-25
-30
40
20
0
-20
-40
-50
0
50
Elevation Angle (degrees)
Azimuth Angle (degrees)
Power (dB)
0
-5
-10
-15
-20
-25
-30

reset

System object: `phased.MVDREstimator2D`
Package: `phased`

Reset states of 2-D MVDR spatial spectrum estimator object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `MVDREstimator2D` object, `H`.

step

System object: `phased.MVDREstimator2D`
Package: `phased`

Perform spatial spectrum estimation

Syntax

```
Y = step(H,X)
[Y,ANG] = step(H,X)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` estimates the spatial spectrum from `X` using the estimator `H`. `X` is a matrix whose columns correspond to channels. `Y` is a matrix representing the magnitude of the estimated 2-D spatial spectrum. The row dimension of `Y` is equal to the number of angles in the `ElevationScanAngles` and the column dimension of `Y` is equal to the number of angles in the `AzimuthScanAngles` property. You can specify the argument, `X`, as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`[Y,ANG] = step(H,X)` returns additional output `ANG` as the signal's direction of arrival (DOA) when the `DOAOutputPort` property is `true`. `ANG` is a two-row matrix where the first row represents estimated azimuth and the second row represents estimated elevation (in degrees).

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Estimate DOA Using 2D MVDR

Estimate the DOAs of two signals received by a 50-element URA with a rectangular lattice. The antenna operating frequency is 150 MHz. The actual direction of the first signal is -37° in azimuth and 0° in elevation. The direction of the second signal is 17° in azimuth and 20° in elevation.

Create signals sampled at 8 kHz.

```
fc = 150e6;
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
array.Element.FrequencyRange = [100e6 300e6];
x = collectPlaneWave(array,[x1 x2],[-37 0;17 20]',fc);
```

Add complex noise.

```
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
```

Create the MVDR DOA estimator for URA.

```
estimator = phased.MVDREstimator2D('SensorArray',array,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
```

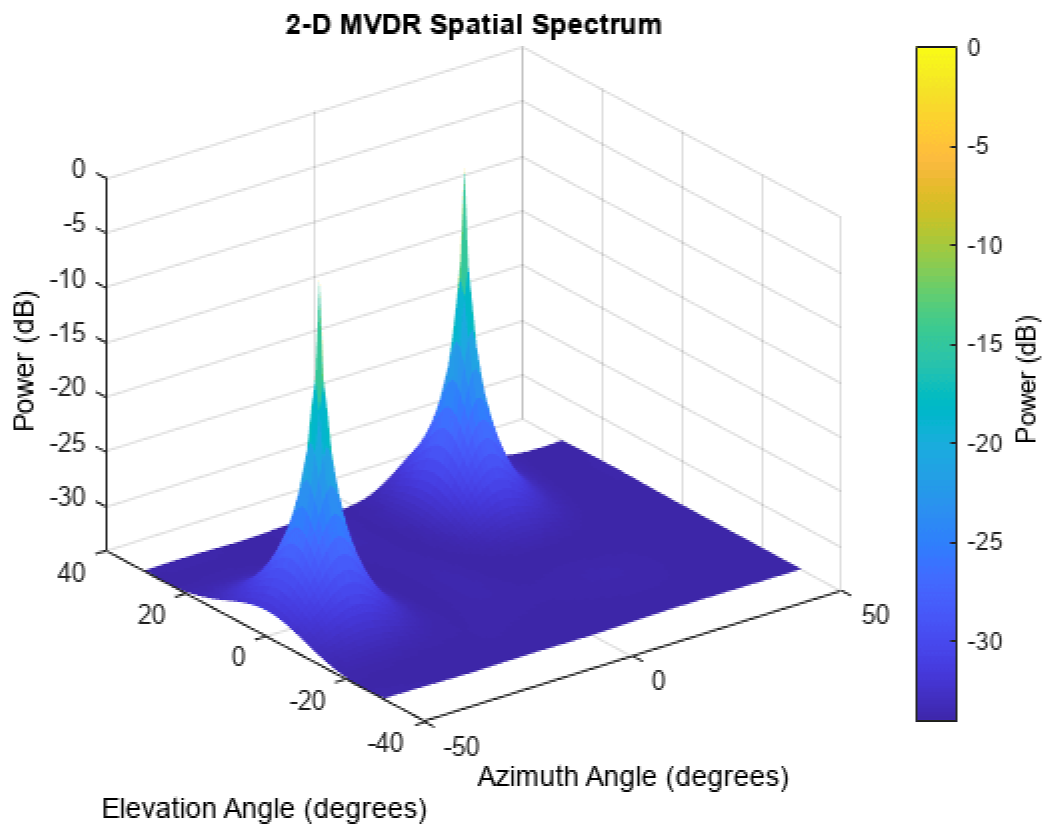
Use the `step` method to the DOA estimates.

```
[~,doas] = estimator(x + noise)
```

```
doas = 2×2

    17   -37
    20     0
```

Plot the spectrum.

```
plotSpectrum(estimator)
```



See Also

`azel2uv` | `azel2phitheta`

phased.MultipathChannel

Package: `phased`

Propagate signals in multipath channel

Description

The `phased.MultipathChannel` System object propagates a signal through a multipath channel. To run the object, you must provide characteristics for each path: time delay, gain, Doppler factor, reflection loss, and spreading loss.

For sonar applications, you can use the `phased.IsoSpeedUnderwaterPaths` System object to generate channel path characteristics. You can also supply these characteristics independently.

To model signal propagation through a multipath channel:

1. Define and set up the propagator. You can set `phased.MultipathChannel` properties at construction time or leave them to their default values. See “Construction” on page 1-1026. Some properties that you set at construction time can be changed later. These properties are *tunable*.
2. To compute the propagated signal, call the `step` method of `phased.MultipathChannel`. The output of the `step` method depends on the properties of the `phased.MultipathChannel` System object. You can change tunable properties at any time.

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`propagator = phased.MultipathChannel` creates a signal propagator System object for a multipath underwater channel.

`propagator = phased.MultipathChannel(Name,Value)` creates a signal propagator System object with each specified property `Name` set to the specified `Value`. You can specify additional name and value pair arguments in any order as (`Name1,Value1`,...,`NameN,ValueN`).

Properties

`OperatingFrequency` — Signal carrier frequency

`20e3` (default) | positive real-valued scalar

Signal carrier frequency, specified as a positive real-valued scalar. Units are in Hz.

Example: `10000`

Data Types: `double`

`SampleRate` — Signal sample rate

`1e3` (default) | positive real-valued scalar

Signal sample rate, specified as a positive real-valued scalar. Units are in Hz. The System object uses this quantity to calculate the propagation delay in units of samples.

Example: `3e3`

Data Types: `double`

`MaximumDelaySource` — Source of maximum delay
`'Auto'` (default) | `'Property'`

Source of the maximum delay value, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the channel automatically allocates enough memory to simulate the propagation delay. When you set this property to `'Property'`, you can specify the maximum delay by using the `MaximumDelay` property. Signals arriving after the maximum delay are ignored.

`MaximumDelay` — Maximum signal delay
`1` (default) | positive scalar

Maximum signal delay, specified as a positive scalar. Delays greater than this value are ignored. Units are in seconds.

Dependencies

To enable this property, set the `MaximumDelaySource` property to `'Property'`.

Data Types: `double`

`InterpolationMethod` — Interpolation method to implement fractional delay
`'Linear'` (default) | `'Oversample'`

Interpolation method used to implement signal fractional delay and Doppler time-dilation and compression, specified as `'Linear'` or `'Oversample'`. When this property is set to `'Linear'`, the input signal is linearly interpolated directly onto a uniform grid to propagate the signal. When this property is set to `'Oversample'`, the input signal is resampled to a higher rate before linear interpolation. For broadband signals, oversampling preserves spectral shape.

Data Types: `char`

Methods

step	Propagate signal through multipath sound channel
reset	Reset state of System object

Common to All System Objects	
`release`	Allow System object property value changes

Examples

One-Way Signal Propagation in Multipath Underwater Sound Channel

Create a five-path underwater sound channel and compute the propagation path matrix, the Doppler factor, and the absorption loss. Assume that the source is stationary and the receiver is moving along the *x*-axis toward the source at 20 km/h. Assume the default one-way propagation.

Create the channel and specify the source and receiver locations and velocities.

```
numpaths = 5;
channel = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',10, ...
    'NumPathsSource','Property','NumPaths',numpaths);
tstep = 1;
srcpos = [0;0;-160];
rcvpos = [100;0;-50];
speed = -20*1000/3600;
srcvel = [0;0;0];
rcvvel = [speed;0;0];
```

Compute the path matrix, Doppler factor, and losses.

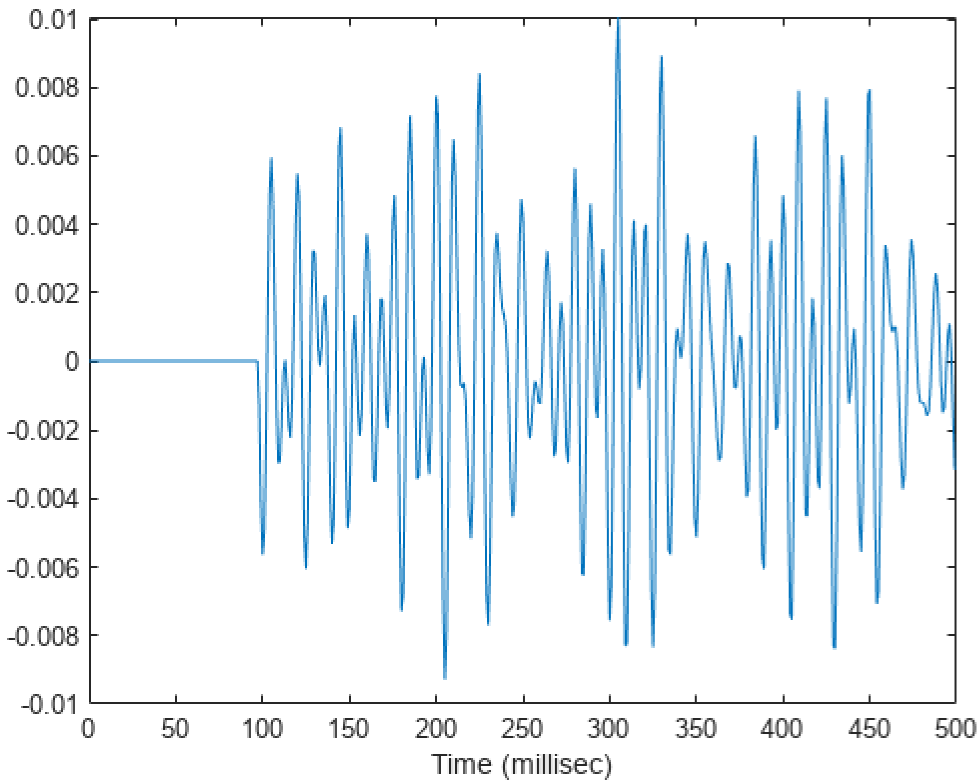
```
[pathmat,dop,absloss] = channel(srcpos,rcvpos,srcvel,rcvvel,tstep);
```

Create 500 samples of a 100 Hz signal. Assume all the paths have the same signal. Propagate the signals to the receiver.

```
fs = 1e3;
nsamp = 500;
propagator = phased.MultipathChannel('OperatingFrequency',10e3,'SampleRate',fs);
t = [0:(nsamp-1)]'/fs;
sig0 = sin(2*pi*100*t);
sig = repmat(sig0,1,numpaths);
propsig = propagator(sig,pathmat,dop,absloss);
```

Plot the real part of the coherent sum of the propagated signals.

```
plot(t*1000,real(sum(propsig,2)))
xlabel('Time (millisec)')
```



Two-Way Signal Propagation in Multipath Underwater Sound Channel

Create a seven-path underwater sound channel and display the propagation path matrix. Assume that the source is stationary and that the receiver is moving along the *x*-axis toward the source at 20 km/h. Assume two-way propagation.

```
speed = -20*1000/3600;
numpaths = 7;
csound = 1515.0;
channel = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200, ...
    'PropagationSpeed',csound,'BottomLoss',10,'NumPathsSource','Property', ...
    'NumPaths',numpaths,'TwoWayPropagation',true);
tstep = 1;
srcpos = [0;0;-160];
tgtpos = [500;0;-50];
srcvel = [0;0;0];
tgtvel = [speed;0;0];
```

Obtain the path matrix, Doppler factor, loss, and target reflection and transmit angles.

```
[pathmat,dop,aloss,tgtangs,srcangs] = channel(srcpos,tgtpos,srcvel,tgtvel,tstep);
```

Create a 100 Hz signal with 500 samples. Assume that all the paths have the same signal but with different amplitudes. Then, propagate the signals to the target and back. You can use the angle information to calculate any angular dependence of the source and target responses. Each channel can have a different amplitude. This example uses a simple cosine model.

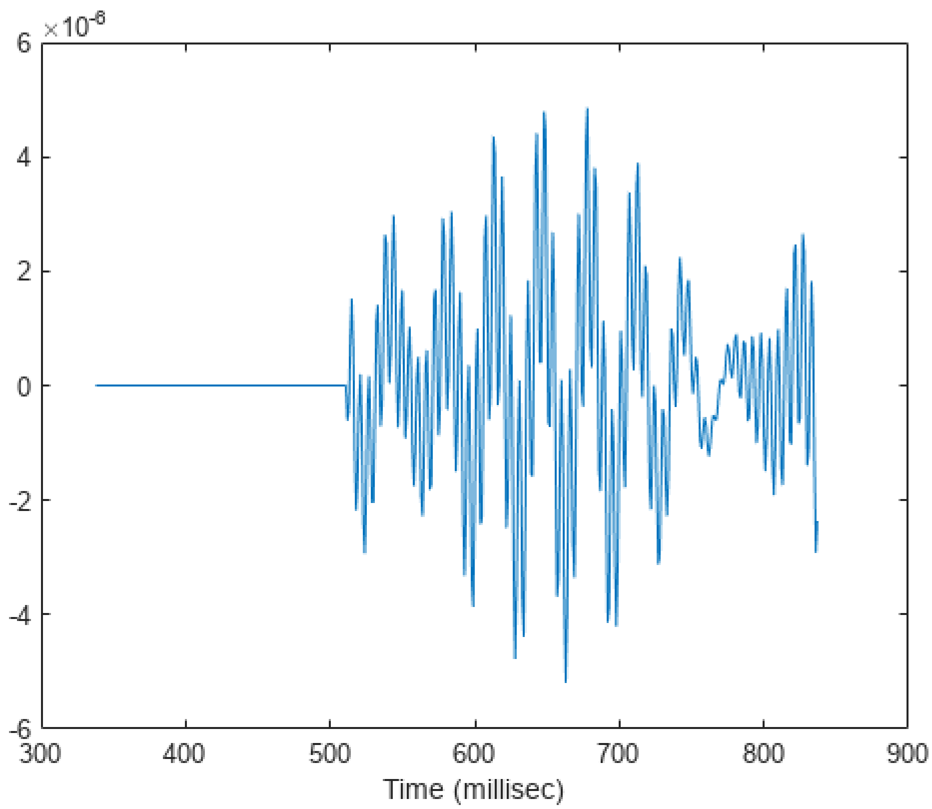
```
fs = 1e3;
nsamp = 500;
propagator = phased.MultipathChannel('OperatingFrequency',10e3,'SampleRate',fs);
t = [0:(nsamp-1)]'/fs;
ampsrc = cosd(srcangs(2,:));
amptgt = cosd(tgtangs(2,:));
sig0 = sin(2*pi*100*t);
sig = repmat(sig0,1,numpaths);
amptotal = ampsrc.^2.*amptgt;
sig = bsxfun(@times,amptotal,sig);
```

Because of the finite propagation delay, the first call to the propagator does not return the signal. Call `propagator` twice to obtain the returned signal.

```
propsig = propagator(sig,pathmat,dop,aloss);
propsig = propagator(sig,pathmat,dop,aloss);
```

Plot the real part of the coherent sum of the propagated signals. Compute the round trip time.

```
rng = rangeangle(srcpos,tgtpos);
tr = rng/csound;
plot((t+tr)*1000,real(sum(propsig,2)))
xlabel('Time (millisec)')
```



Propagate Sound in Channel Having Unknown Number of Paths

Create an underwater sound channel and plot the combined received signal. Automatically find the number of paths. Assume that the source is stationary and that the receiver is moving along the *x*-axis toward the source at 20 km/h. Assume the default one-way propagation.

```
speed = -20*1000/3600;
channel = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',5, ...
    'NumPathsSource','Auto','CoherenceTime',5);
tstep = 1;
srcpos = [0;0;-160];
rcvpos = [500;0;-50];
srcvel = [0;0;0];
rcvvel = [speed;0;0];
```

Compute the path matrix, Doppler factor, and losses. The propagator outputs 51 paths output but some paths can contain `Nan` values.

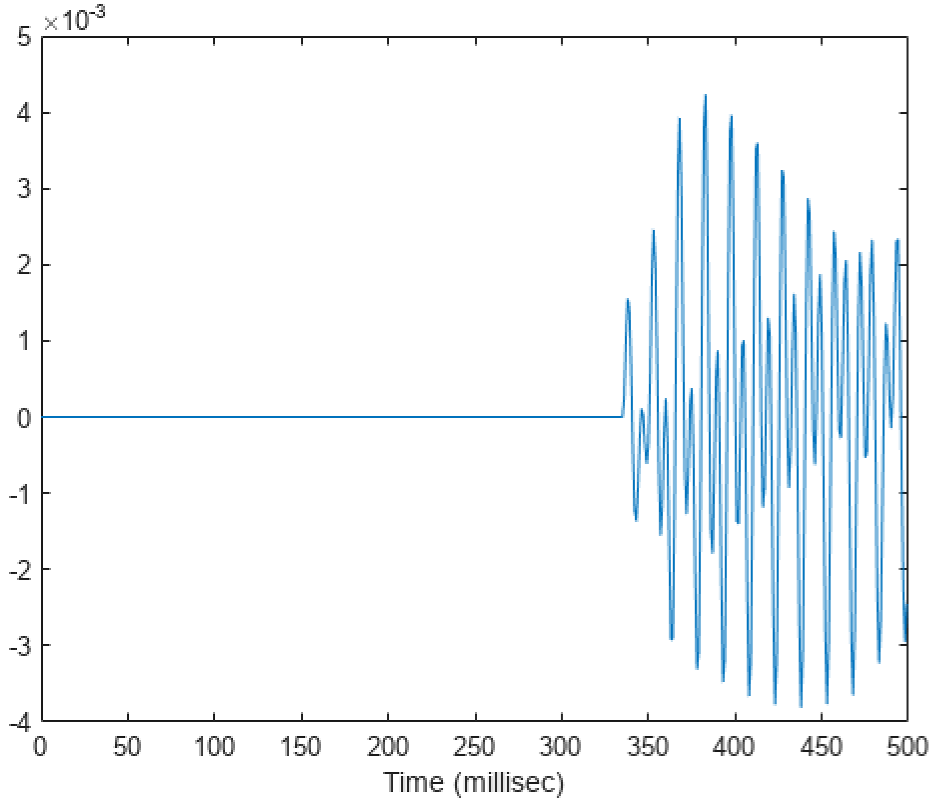
```
[pathmat,dop,absloss,rcvangs,srcangs] = channel(srcpos,rcvpos,srcvel,rcvvel,tstep);
```

Create of a 100 Hz signal with 500 samples. Assume that all the paths have the same signal. Use a `phased.MultipathChannel` System object™ to propagate the signals to the receiver. `phased.MultipathChannel` accepts as input all paths produced by `phased.IsoSpeedUnderwaterPaths` but ignores paths that have `NaN` values.

```
fs = 1e3;
nsamp = 500;
propagator = phased.MultipathChannel('OperatingFrequency',10e3,'SampleRate',fs);
t = [0:(nsamp-1)]'/fs;
sig0 = sin(2*pi*100*t);
numpaths = size(pathmat,2);
sig = repmat(sig0,1,numpaths);
propsig = propagator(sig,pathmat,dop,absloss);
```

Plot the real part of the coherent sum of the propagated signals.

```
plot(t*1000,real(sum(propsig,2)))
xlabel('Time (millisec)')
```



Doppler Stretching of Sonar Signal

Compare the duration of a propagated signal from a stationary sonar to that of a moving sonar. The moving sonar has a radial velocity of 25 m/s away from the target. In each case, propagate the signal along a single path. Assume one-way propagation.

Define the sonar system parameters: maximum unambiguous range, required range resolution, operating frequency, and propagation speed.

```
maxrange = 5000.0;
rngres = 10.0;
fc = 20.0e3;
csound = 1520.0;
```

Use a rectangular waveform for the transmitted signal.

```
prf = csound/(2*maxrange);
pulseWidth = 8*rngres/csound;
pulseBW = 1/pulseWidth;
fs = 80*pulseBW;
waveform = phased.RectangularWaveform('PulseWidth',pulseWidth,'PRF',prf, ...
    'SampleRate',fs);
```

Specify the sonar positions.

```
sonarplatform1 = phased.Platform('InitialPosition',[0;0;-60],'Velocity',[0;0;0]);
sonarplatform2 = phased.Platform('InitialPosition',[0;0;-60],'Velocity',[0;-25;0]);
```

Specify the target position.

```
targetplatform = phased.Platform('InitialPosition',[0;500;-60],'Velocity',[0;0;0]);
```

Define the underwater path and propagation channel objects.

```
paths = phased.IsoSpeedUnderwaterPaths('ChannelDepth',100, ...
    'CoherenceTime',0,'NumPathsSource','Property','NumPaths',1, ...
    'PropagationSpeed',csound);
propagator = phased.MultipathChannel('SampleRate',fs,'OperatingFrequency',fc);
```

Create the transmitted waveform.

```
wav = waveform();
nsamp = size(wav,1);
rxpulses = zeros(nsamp,2);
t = (0:nsamp-1)/fs;
```

Transmit the signal and then receive the echo at the stationary sonar.

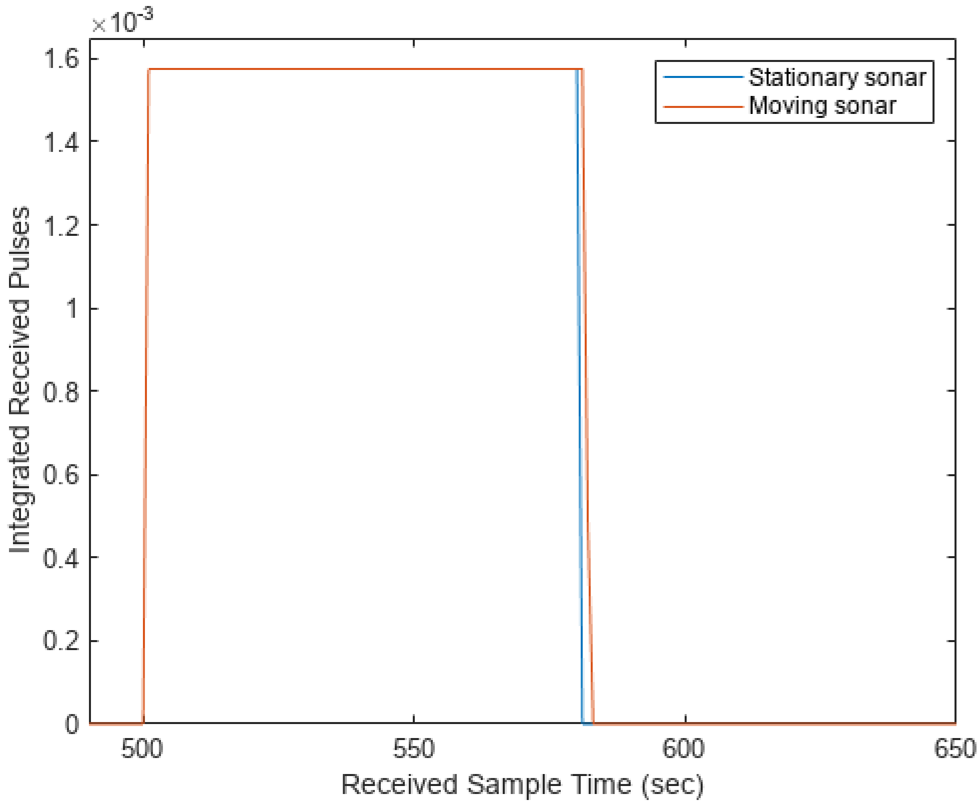
```
[pathmat,dop,aloss,~,~] = paths(sonarplatform1.InitialPosition, ...
    targetplatform.InitialPosition,sonarplatform1.InitialVelocity, ...
    targetplatform.InitialVelocity,1/prf);
rxpulses(:,1) = propagator(wav,pathmat,dop,aloss);
```

Transmit and receive at the moving sonar.

```
[pathmat,dop,aloss,~,~] = paths(sonarplatform2.InitialPosition, ...
    targetplatform.InitialPosition,sonarplatform2.Velocity, ...
    targetplatform.Velocity,1/prf);
rxpulses(:,2) = propagator(wav,pathmat,dop,aloss);
```

Plot the received pulses.

```
plot(abs(rxpulses))
xlim([490 650])
ylim([0 1.65e-3])
legend('Stationary sonar','Moving sonar')
xlabel('Received Sample Time (sec)')
ylabel('Integrated Received Pulses')
```



The signal received at the moving sonar has increased in duration compared to the stationary sonar.

Version History

Introduced in R2017a

References

[1] Urick, R.J. *Principles of Underwater Sound, 3rd Edition*. New York: Peninsula Publishing, 1996.

[2] Sherman, C.S. and J. Butler *Transducers and Arrays for Underwater Sound*. New York: Springer, 2007.

[3] Allen, J.B. and D. Berkely, "Image method for efficiently simulating small-room acoustics", J. Acoust. Soc. Am, Vol 65, No. 4. April 1979.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects
`phased.BackscatterSonarTarget` | `phased.IsoSpeedUnderwaterPaths` | `phased.IsotropicHydrophone` | `phased.IsotropicProjector`

Topics
"Underwater Target Detection with an Active Sonar System"
"Locating an Acoustic Beacon with a Passive Sonar System"
"Doppler Effect for Sound"

step

System object: `phased.MultipathChannel`
Package: `phased`

Propagate signal through multipath sound channel

Syntax

`propsig = step(propagator,sig,pathmat,dop,aloss)`

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`propsig = step(propagator,sig,pathmat,dop,aloss)` returns a signal, `propsig`, propagated through a multipath channel. `sig` is the input signal to the channel. The `pathmat` matrix contains the path time delay, the total reflection coefficient, and the spreading loss. `dop` specifies the Doppler factor and `aloss` specifies the frequency-dependent absorption loss. The matrix can describe one-way or two-way propagation from the signal source position to the signal destination position.

- When you use this method for one-way propagation, the source refers to the origin of the signal and the destination refers to the receiver. You can use one-way propagation modeling to model passive sonar and underwater communications.
- When you use this method for two-way propagation, the destination refers to the reflecting target, not the sonar receiver. A two-way path consists of a two identical one-way paths from source to target and back to receiver (collocated with the source). You can use two-way propagation to model active sonar systems.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Input Arguments

`propagator` — Multipath channel propagator
`phased.MultipathChannel` System object

Multipath channel propagator, specified as a `phased.MultipathChannel` System object.

Example: `phased.MultipathChannel`

`sig` — Channel input signal
complex-valued *M*-by-*N* matrix

Channel input signal, specified as a complex-valued *M*-by-*N* matrix. *M* is the number of samples in the signal and *N* is the number of paths.

Data Types: `double`

`pathmat` — Propagation paths matrix
real-valued 3-by-*N* matrix

Propagation paths matrix, specified as a real-valued 3-by-*N* matrix. *N* is the number of paths in the channel. Each column represents a path. The matrix rows represent:

Row	Data
1	Propagation delays for each path. Units are in seconds.
2	Total reflection coefficient for each path. Units are dimensionless
3	Spreading loss for each path. Units are in dB.

Except for the direct path, paths consist of alternating surface and bottom reflections. The losses for multiple reflections at the boundaries are multiplied. When you use `phased.IsoSpeedUnderwaterPaths` to create a path matrix, some of the columns can contain `NaN` values. `phased.MultipathChannel` ignores these paths.

Data Types: `double`

`dop` — Doppler factor
real-valued *N*-by-1 row vector

Doppler factor, specified as a real-valued *N*-by-1 row vector where *N* is the number of paths. The Doppler factor multiplies the transmitted frequency to produce the Doppler-shifted frequency for each path. The factor also defines the time contraction or dilation of a signal. Units are dimensionless.

Data Types: `double`

`aloss` — Frequency-dependent absorption loss
real-valued *K*-by-*N*+1 matrix

Frequency-dependent absorption loss, specified as a real-valued *K*-by-*N*+1 matrix. *K* is the number of frequencies and *N* is the number of paths. The first column of `aloss` contains the absorption-loss frequencies in Hz. The remaining columns contain the absorption losses for the corresponding frequency. Units are in dB.

Data Types: `double`

Output Arguments

`propsig` — Channel output signal
complex-valued *M*-by-*N* matrix

Channel output signal, returned as a complex-valued *M*-by-*N* matrix. *M* is the number of samples in the signal and *N* is the number of paths. The output is the signal propagated through the channel. `propsig` has the same dimensions as the input signal, `sig`.

Examples

One-Way Signal Propagation in Multipath Underwater Sound Channel

Create a five-path underwater sound channel and compute the propagation path matrix, the Doppler factor, and the absorption loss. Assume that the source is stationary and the receiver is moving along the *x*-axis toward the source at 20 km/h. Assume the default one-way propagation.

Create the channel and specify the source and receiver locations and velocities.

```
numpaths = 5;
channel = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',10, ...
    'NumPathsSource','Property','NumPaths',numpaths);
tstep = 1;
srcpos = [0;0;-160];
rcvpos = [100;0;-50];
speed = -20*1000/3600;
srcvel = [0;0;0];
rcvvel = [speed;0;0];
```

Compute the path matrix, Doppler factor, and losses.

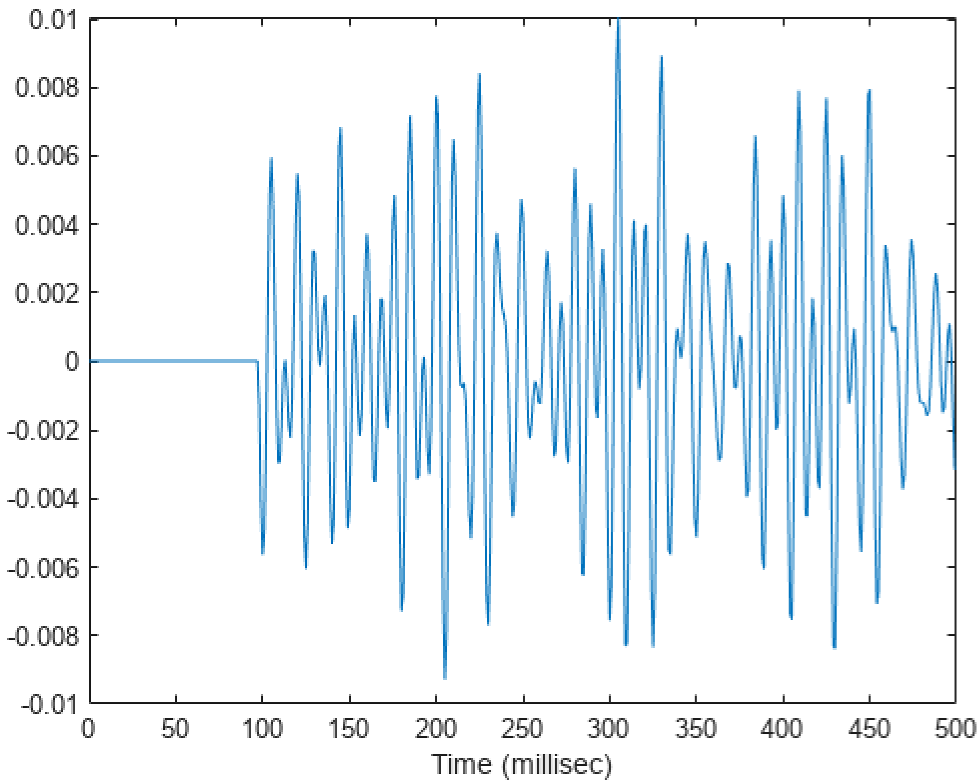
```
[pathmat,dop,absloss] = channel(srcpos,rcvpos,srcvel,rcvvel,tstep);
```

Create 500 samples of a 100 Hz signal. Assume all the paths have the same signal. Propagate the signals to the receiver.

```
fs = 1e3;
nsamp = 500;
propagator = phased.MultipathChannel('OperatingFrequency',10e3,'SampleRate',fs);
t = [0:(nsamp-1)]'/fs;
sig0 = sin(2*pi*100*t);
sig = repmat(sig0,1,numpaths);
propsig = propagator(sig,pathmat,dop,absloss);
```

Plot the real part of the coherent sum of the propagated signals.

```
plot(t*1000,real(sum(propsig,2)))
xlabel('Time (millisec)')
```



Two-Way Signal Propagation in Multipath Underwater Sound Channel

Create a seven-path underwater sound channel and display the propagation path matrix. Assume that the source is stationary and that the receiver is moving along the *x*-axis toward the source at 20 km/h. Assume two-way propagation.

```
speed = -20*1000/3600;
numpaths = 7;
csound = 1515.0;
channel = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200, ...
    'PropagationSpeed',csound,'BottomLoss',10,'NumPathsSource','Property', ...
    'NumPaths',numpaths,'TwoWayPropagation',true);
tstep = 1;
srcpos = [0;0;-160];
tgtpos = [500;0;-50];
srcvel = [0;0;0];
tgtvel = [speed;0;0];
```

Obtain the path matrix, Doppler factor, loss, and target reflection and transmit angles.

```
[pathmat,dop,aloss,tgtangs,srcangs] = channel(srcpos,tgtpos,srcvel,tgtvel,tstep);
```

Create a 100 Hz signal with 500 samples. Assume that all the paths have the same signal but with different amplitudes. Then, propagate the signals to the target and back. You can use the angle information to calculate any angular dependence of the source and target responses. Each channel can have a different amplitude. This example uses a simple cosine model.

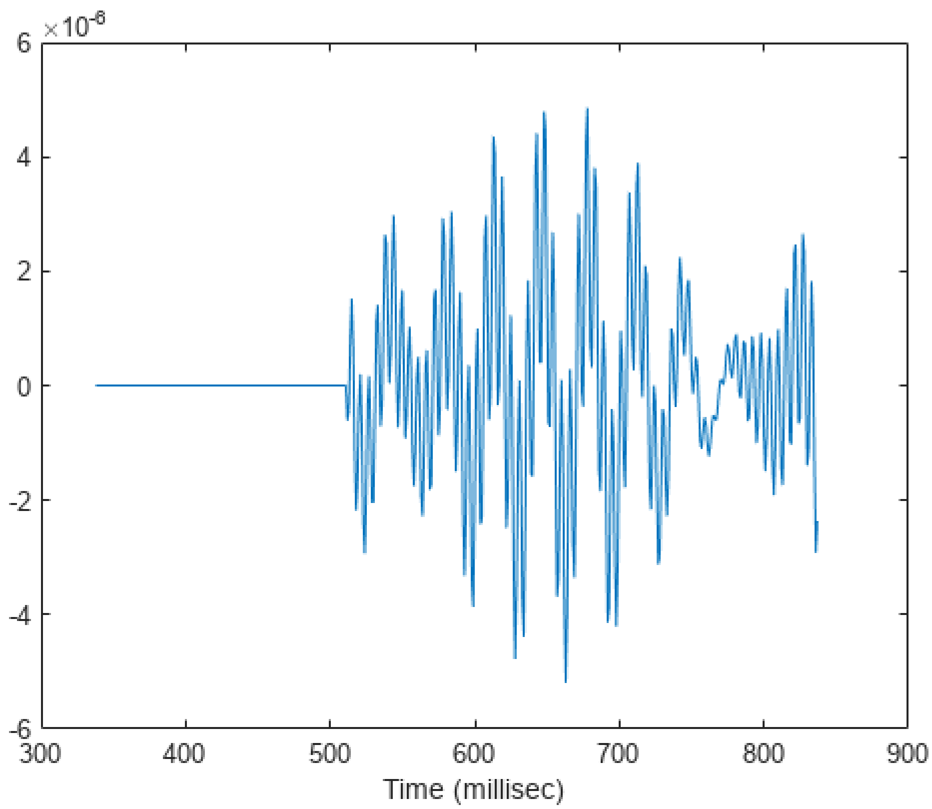
```
fs = 1e3;
nsamp = 500;
propagator = phased.MultipathChannel('OperatingFrequency',10e3,'SampleRate',fs);
t = [0:(nsamp-1)]'/fs;
ampsrc = cosd(srcangs(2,:));
amptgt = cosd(tgtangs(2,:));
sig0 = sin(2*pi*100*t);
sig = repmat(sig0,1,numpaths);
amptotal = ampsrc.^2.*amptgt;
sig = bsxfun(@times,amptotal,sig);
```

Because of the finite propagation delay, the first call to the propagator does not return the signal. Call `propagator` twice to obtain the returned signal.

```
propsig = propagator(sig,pathmat,dop,aloss);
propsig = propagator(sig,pathmat,dop,aloss);
```

Plot the real part of the coherent sum of the propagated signals. Compute the round trip time.

```
rng = rangeangle(srcpos,tgtpos);
tr = rng/csound;
plot((t+tr)*1000,real(sum(propsig,2)))
xlabel('Time (millisec)')
```



Propagate Sound in Channel Having Unknown Number of Paths

Create an underwater sound channel and plot the combined received signal. Automatically find the number of paths. Assume that the source is stationary and that the receiver is moving along the *x*-axis toward the source at 20 km/h. Assume the default one-way propagation.

```
speed = -20*1000/3600;
channel = phased.IsoSpeedUnderwaterPaths('ChannelDepth',200,'BottomLoss',5, ...
    'NumPathsSource','Auto','CoherenceTime',5);
tstep = 1;
srcpos = [0;0;-160];
rcvpos = [500;0;-50];
srcvel = [0;0;0];
rcvvel = [speed;0;0];
```

Compute the path matrix, Doppler factor, and losses. The propagator outputs 51 paths output but some paths can contain `Nan` values.

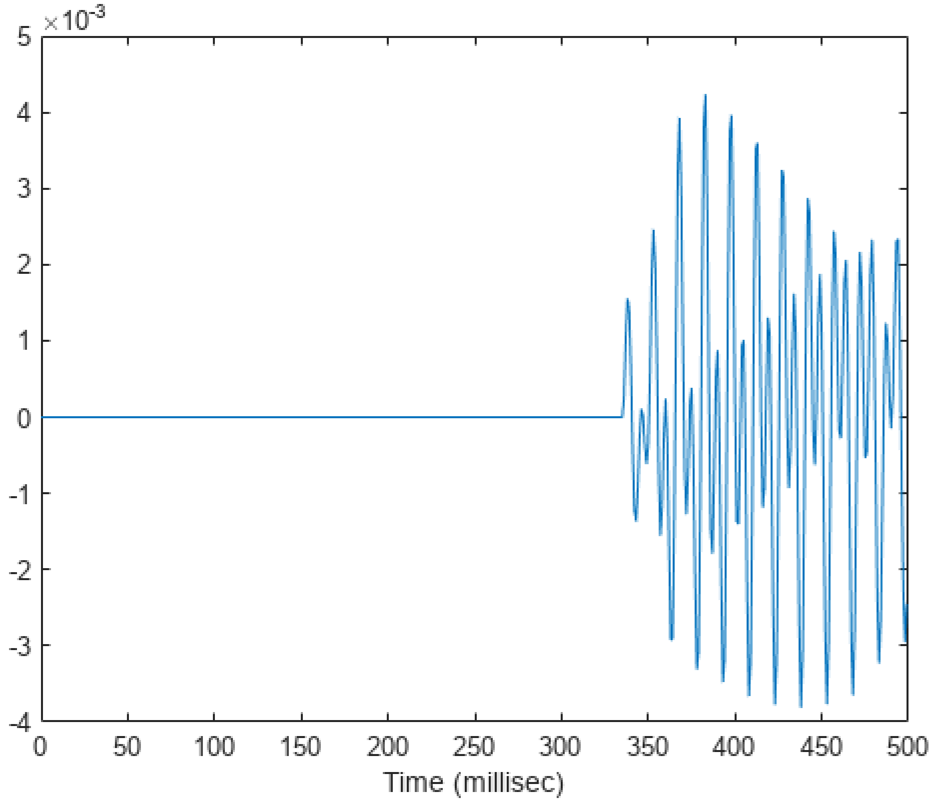
```
[pathmat,dop,absloss,rcvangs,srcangs] = channel(srcpos,rcvpos,srcvel,rcvvel,tstep);
```

Create of a 100 Hz signal with 500 samples. Assume that all the paths have the same signal. Use a `phased.MultipathChannel` System object™ to propagate the signals to the receiver. `phased.MultipathChannel` accepts as input all paths produced by `phased.IsoSpeedUnderwaterPaths` but ignores paths that have `NaN` values.

```
fs = 1e3;
nsamp = 500;
propagator = phased.MultipathChannel('OperatingFrequency',10e3,'SampleRate',fs);
t = [0:(nsamp-1)]'/fs;
sig0 = sin(2*pi*100*t);
numpaths = size(pathmat,2);
sig = repmat(sig0,1,numpaths);
propsig = propagator(sig,pathmat,dop,absloss);
```

Plot the real part of the coherent sum of the propagated signals.

```
plot(t*1000,real(sum(propsig,2)))
xlabel('Time (millisec)')
```



Version History

Introduced in R2017a

reset

System object: `phased.MultipathChannel`
Package: `phased`

Reset state of System object

Syntax

`reset(propagator)`

Description

`reset(propagator)` resets the internal state of the `phased.MultipathChannel` object, `propagator`.

Input Arguments

`propagator` — Multipath channel
`phased.MultipathChannel` System object

Multipath channel, specified as a `phased.MultipathChannel` System object.

Example: `phased.MultipathChannel`

Version History

Introduced in R2017a

phased.MUSICEstimator

Package: phased

Estimate direction of arrival using narrowband MUSIC algorithm for ULA

Description

The `phased.MUSICEstimator` System object implements the narrowband multiple signal classification (MUSIC) algorithm for uniform linear arrays (ULA). MUSIC is a high-resolution direction-finding algorithm capable of resolving closely-spaced signal sources. The algorithm is based on eigenspace decomposition of the sensor spatial covariance matrix.

To estimate directions of arrival (DOA):

1. Define and set up a `phased.MUSICEstimator` System object. See “Construction” on page 1-1044.
2. Call the `step` method to estimate the DOAs according to the properties of `phased.MUSICEstimator`.

Note Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`estimator = phased.MUSICEstimator` creates a MUSIC DOA estimator System object, `estimator`.

`estimator = phased.MUSICEstimator(Name,Value)` creates a System object, `estimator`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray` — ULA sensor array
`phased.ULA` System object (default)

ULA sensor array, specified as a `phased.ULA` System object. If you do not specify any name-value pair properties for the ULA sensor array, the default properties of the array are used.

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`.

Example: `3e8`

Data Types: `single` | `double`

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: 1e9

Data Types: single | double

ForwardBackwardAveraging — Enable forward-backward averaging
false (default) | true

Enable forward-backward averaging, specified as false or true. Set this property to true to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold.

Data Types: logical

ScanAngles — Broadside scan angles
[-90:90] (default) | real-valued *K*-length vector

Broadside scan angles, specified as a real-valued vector. Units are in degrees. Broadside angles are between the search direction and the ULA array axis. The angles lie between –90° and 90°, inclusive. Specify the angles in increasing value.

Example: [-20:20]

Data Types: single | double

DOAOutputPort — Enable directions of arrival output
false (default) | true

Option to enable directions-of-arrival (DOA) output, specified as false or true. To obtain the DOA of signals, set this property to true. The DOAs are returned in the second output argument when the object is executed.

Data Types: logical

NumSignalsSource — Source of number of signals
'Auto' (default) | 'Property'

Source of the number of arriving signals, specified as 'Auto' or 'Property'.

- 'Auto' — The System object estimates the number of arriving signals using the method specified in the NumSignalsMethod property.
- 'Property' — Specify the number of arriving signals using the NumSignals property.

Data Types: char

NumSignalsMethod — Method used to estimate number of arriving signals
'AIC' (default) | 'MDL'

Method used to estimate the number of arriving signals, specified as 'AIC' or 'MDL'.

- 'AIC' — Akaike Information Criterion
- 'MDL' — Minimum Description Length criterion

Dependencies

To enable this property, set `NumSignalsSource` to `'Auto'`.

Data Types: `char`

`NumSignals` — Number of arriving signals
1 (default) | positive integer

Number of arriving signals for DOA estimation, specified as a positive integer.

Example: 3

Dependencies

To enable this property, set `NumSignalsSource` to `'Property'`.

Data Types: `single` | `double`

`SpatialSmoothing` — Enable spatial smoothing
0 (default) | nonnegative integer

Option to enable spatial smoothing, specified as a nonnegative integer. Use spatial smoothing to compute the arrival directions of coherent signals. A value of zero specifies no spatial smoothing. A positive value represents the number of subarrays used to compute the smoothed (averaged) source covariance matrix. Each increment in this value lets you handle one additional coherent source, but reduces the effective number of array elements by one. The length of the smoothing aperture, L, depends on the array length, M, and the averaging number, K, by $L = M - K + 1$. The maximum value of K is $M - 2$.

Example: 5

Data Types: `double`

Methods

plotSpectrum	Plot MUSIC spectrum
reset	Reset states of System object
step	Estimate direction of arrival using MUSIC

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Plot MUSIC Spectrum of Two Signals Arriving at ULA

Estimate the DOAs of two signals received by a standard 10-element ULA having an element spacing of 1 meter. Then plot the MUSIC spectrum.

Create the ULA array. The antenna operating frequency is 150 MHz.

```
fc = 150.0e6;
array = phased.ULA('NumElements',10,'ElementSpacing',1.0);
```

Create the arriving signals at the ULA. The true direction of arrival of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 60° in azimuth and -5° in elevation.

```
fs = 8000.0;
t = (0:1/fs:1).';
sig1 = cos(2*pi*t*300.0);
sig2 = cos(2*pi*t*400.0);
sig = collectPlaneWave(array,[sig1 sig2],[10 20; 60 -5]',fc);
noise = 0.1*(randn(size(sig)) + 1i*randn(size(sig)));
```

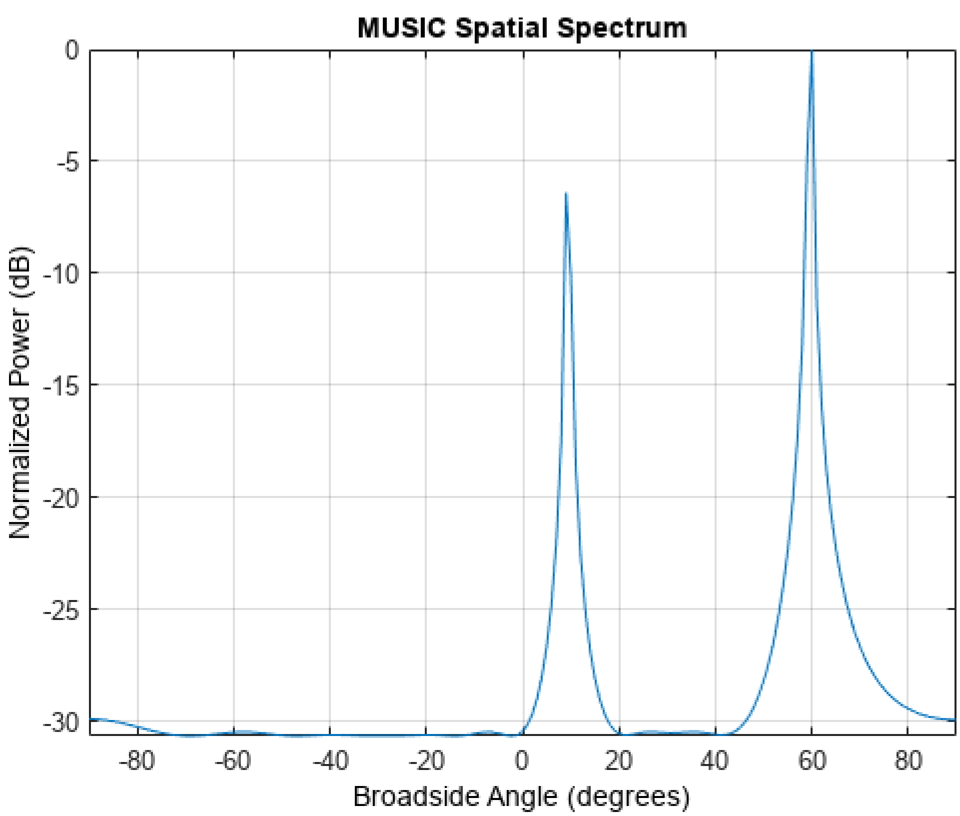
Estimate the DOAs.

```
estimator = phased.MUSICEstimator('SensorArray',array,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignalsSource','Property',...
    'NumSignals',2);
[y,doas] = estimator(sig + noise);
doas = broadside2az(sort(doas),[20 -5])
```

```
doas = 1×2

    9.5829   60.3813
```

Plot the MUSIC spectrum.

```
plotSpectrum(estimator,'NormalizeResponse',true)
```



Compute DOA of Two Nearby Signals Using MUSIC

First, estimate the DOAs of two signals received by a standard 10-element ULA having an element spacing of one-half wavelength.Then, plot the spatial spectrum.

The antenna operating frequency is 150 MHz. The arrival directions of the two signals are separated by 2°. The direction of the first signal is 30° azimuth and 0° elevation. The direction of the second signal is 32° azimuth and 0° elevation. Estimate the number of signals using the Minimum Description Length (MDL) criterion.

Create the signals arriving at the ULA.

```
fs = 8000;
t = (0:1/fs:1).';
f1 = 300.0;
f2 = 600.0;
sig1 = cos(2*pi*t*f1);
sig2 = cos(2*pi*t*f2);
fc = 150.0e6;
c = physconst('LightSpeed');
lam = c/fc;
array = phased.ULA('NumElements',10,'ElementSpacing',0.5*lam);
sig = collectPlaneWave(array,[sig1 sig2],[30 0; 32 0]',fc);
noise = 0.1*(randn(size(sig)) + 1i*randn(size(sig)));
```

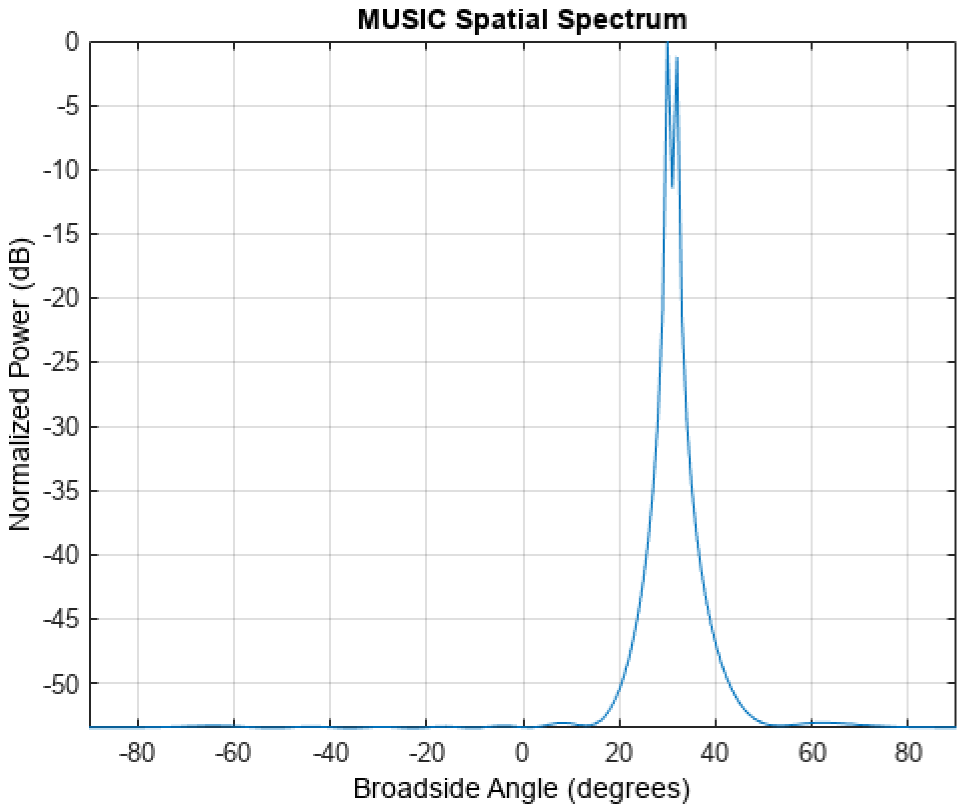
Estimate the DOAs.

```
estimator = phased.MUSICEstimator('SensorArray',array,...
    'OperatingFrequency',fc,'DOAOutputPort',true,...
    'NumSignalsSource','Auto','NumSignalsMethod','MDL');
[y,doas] = estimator(sig + noise);
doas = broadside2az(sort(doas),[0 0])
```

```
doas = 1×2

   30.0000   32.0000
```

Plot the MUSIC spectrum.

```
plotSpectrum(estimator,'NormalizeResponse',true)
```



Algorithms

MUSIC Algorithm

MUSIC is a high-resolution direction-finding algorithm that estimates directions of arrival (DOA) of signals at an array from the covariance matrix of array sensor data. MUSIC belongs to the subspace-decomposition family of direction-finding algorithms. Unlike conventional beamforming, MUSIC can resolve closely spaced signal sources.

Based on eigenspace decomposition of the sensor covariance matrix, MUSIC divides the observation space into orthogonal signal and noise subspaces. Eigenvectors corresponding to the largest eigenvalues span the signal subspace. Eigenvectors corresponding to the smaller eigenvalues span the noise subspace. Because arrival (or steering) vectors lie in the signal subspace, they are orthogonal to the noise subspace. For ULAs, arrival vectors are functions of the broadside direction angles of the sources. The algorithm searches a grid of arrival angles to find the arrival vectors that have zero or small projections into the noise subspace. These angles are the directions of the sources.

MUSIC requires that the number of source signals is known. If the number of specified sources does not match the actual number of sources, the algorithm degrades. Generally, you must provide an estimate of the number of sources or use one of the built-in source number estimation methods. For a description of the methods used to estimate the number of sources, see the `aictest` or `mdltest` functions.

In place of the true sensor covariance matrix, the algorithm computes the sample covariance matrix from the sensor data. MUSIC applies to noncoherent signals but can be extended to coherent signals using spatial smoothing and/or forward-backward averaging techniques. For a high-level description of the algorithm, see “MUSIC Super-Resolution DOA Estimation”.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2016b

References

[1] Van Trees, H. L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

Functions

`aictest` | `broadside2az` | `mdltest` | `rootmusicdoa` | `spsmooth`

Objects

`phased.MUSICEstimator2D` | `phased.RootMUSICEstimator`

Topics

“MUSIC Super-Resolution DOA Estimation”
“Direction of Arrival Estimation with Beamscan, MVDR, and MUSIC”
“High Resolution Direction of Arrival Estimation”
“Spherical Coordinates”

plotSpectrum

System object: phased.MUSICEstimator
Package: phased

Plot MUSIC spectrum

Syntax

```
plotSpectrum(estimator)
output_args = plotSpectrum(estimator,Name,Value)
lh = plotSpectrum( ___ )
```

Description

plotSpectrum(estimator) plots the MUSIC spectrum computed by the most recent step method execution for the phased.MUSICEstimator System object, estimator.

output_args = plotSpectrum(estimator,Name,Value) plots the MUSIC spatial spectrum with additional options specified by one or more Name,Value pair arguments.

lh = plotSpectrum(___) returns the line handle to the figure.

Input Arguments

estimator — MUSIC estimator
phased.MUSICEstimator System object.

MUSIC estimator, specified as a phased.MUSICEstimator System object

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Unit — Units used for plotting
'db' (default) | 'mag' | 'pow'

Units used for plotting, specified as the comma-separated pair consisting of 'Unit' and 'db', 'mag', or 'pow'.

Data Types: char

NormalizeResponse — Plot normalized spectrum
false (default) | true

Plot a normalized spectrum, specified as the comma-separated pair consisting of 'NormalizedResponse' and false or true. Normalization sets the magnitude of the largest spectrum value to one.

Data Types: `char`

`Title` — Title of plot
`'MUSIC Spatial Spectrum'` (default) | character vector

Title of plot, specified as a comma-separated pair consisting of `'Title'` and a character vector.

Example: `true`

Data Types: `char`

Output Arguments

`lh` — Line handle of plot
line handle

Line handle of plot.

Examples

Plot MUSIC Spectrum of Two Signals Arriving at ULA

Estimate the DOAs of two signals received by a standard 10-element ULA having an element spacing of 1 meter. Then plot the MUSIC spectrum.

Create the ULA array. The antenna operating frequency is 150 MHz.

```
fc = 150.0e6;
array = phased.ULA('NumElements',10,'ElementSpacing',1.0);
```

Create the arriving signals at the ULA. The true direction of arrival of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 60° in azimuth and -5° in elevation.

```
fs = 8000.0;
t = (0:1/fs:1).';
sig1 = cos(2*pi*t*300.0);
sig2 = cos(2*pi*t*400.0);
sig = collectPlaneWave(array,[sig1 sig2],[10 20; 60 -5]',fc);
noise = 0.1*(randn(size(sig)) + 1i*randn(size(sig)));
```

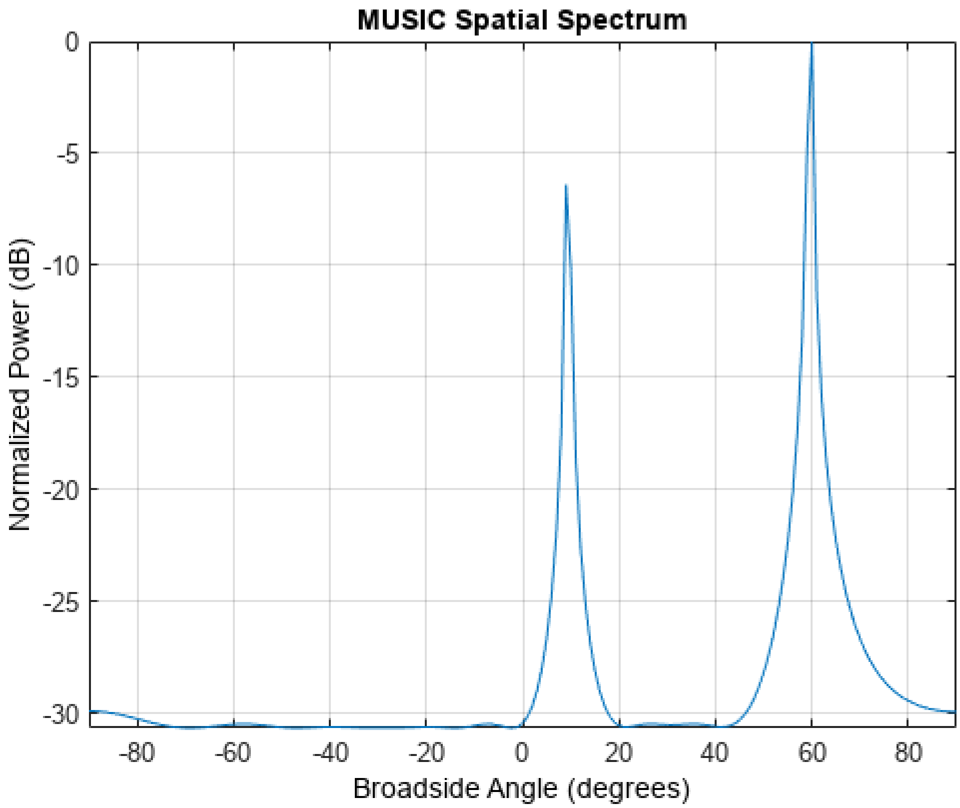
Estimate the DOAs.

```
estimator = phased.MUSICEstimator('SensorArray',array,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignalsSource','Property',...
    'NumSignals',2);
[y,doas] = estimator(sig + noise);
doas = broadside2az(sort(doas),[20 -5])
```

```
doas = 1×2

    9.5829   60.3813
```

Plot the MUSIC spectrum.

```
plotSpectrum(estimator,'NormalizeResponse',true)
```



Version History

Introduced in R2016b

reset

System object: `phased.MUSICEstimator`
Package: `phased`

Reset states of System object

Syntax

`reset(estimator)`

Description

`reset(estimator)` resets the internal state of the `phased.MUSICEstimator` System object, `estimator`.

Input Arguments

`estimator` — MUSIC estimator
`phased.MUSICEstimator` System object

MUSIC estimator, specified as a `phased.MUSICEstimator` System object.

Version History

Introduced in R2016b

step

System object: `phased.MUSICEstimator`
Package: `phased`

Estimate direction of arrival using MUSIC

Syntax

```
spectrum = step(estimator,X)
[spectrum,doa] = step(estimator,X)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`spectrum = step(estimator,X)` returns the MUSIC spectrum for a signal specified by `X`.

`[spectrum,doa] = step(estimator,X)` also returns the signal broadside directions of arrival, `doa`. To use this syntax, set the `DOAOutputPort` property to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`estimator` — MUSIC estimator

`phased.MUSICEstimator` System object

MUSIC estimator, specified as a `phased.MUSICEstimator` System object.

Example: `phased.MUSICEstimator`

X — Received signal

M-by-*N* complex-valued matrix

Received signal, specified as an *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values (snapshots) contained in the signal, and *N* is the number of sensor elements in the array.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[[0;1;2;3;4;3;2;1;0],[1;2;3;4;3;2;1;0;0]]`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`spectrum` — MUSIC spatial spectrum

nonnegative, real-valued *K*-length column vector

MUSIC spatial spectrum, returned as a non-negative, real-valued *K*-length column vector representing the magnitude of the estimated MUSIC spatial spectrum. Each entry corresponds to an angle specified by the `ScanAngles` property.

doa — Directions of arrival

real-valued *L*-length row vector

Directions of arrival of the signals, returned as a real-valued *L*-length row vector. The direction of arrival angle is the angle between the source direction and the array axis or broadside angle. Angle units are in degrees. *L* is the number of signals specified by the `NumSignals` property or computed using the method specified by the `NumSignalsMethod` property.

Dependencies

To enable this output argument, set the `DOAOutputPort` property to `true`.

Examples

Plot MUSIC Spectrum of Two Signals Arriving at ULA

Estimate the DOAs of two signals received by a standard 10-element ULA having an element spacing of 1 meter. Then plot the MUSIC spectrum.

Create the ULA array. The antenna operating frequency is 150 MHz.

```
fc = 150.0e6;
array = phased.ULA('NumElements',10,'ElementSpacing',1.0);
```

Create the arriving signals at the ULA. The true direction of arrival of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 60° in azimuth and -5° in elevation.

```
fs = 8000.0;
t = (0:1/fs:1).';
sig1 = cos(2*pi*t*300.0);
sig2 = cos(2*pi*t*400.0);
sig = collectPlaneWave(array,[sig1 sig2],[10 20; 60 -5]',fc);
noise = 0.1*(randn(size(sig)) + 1i*randn(size(sig)));
```

Estimate the DOAs.

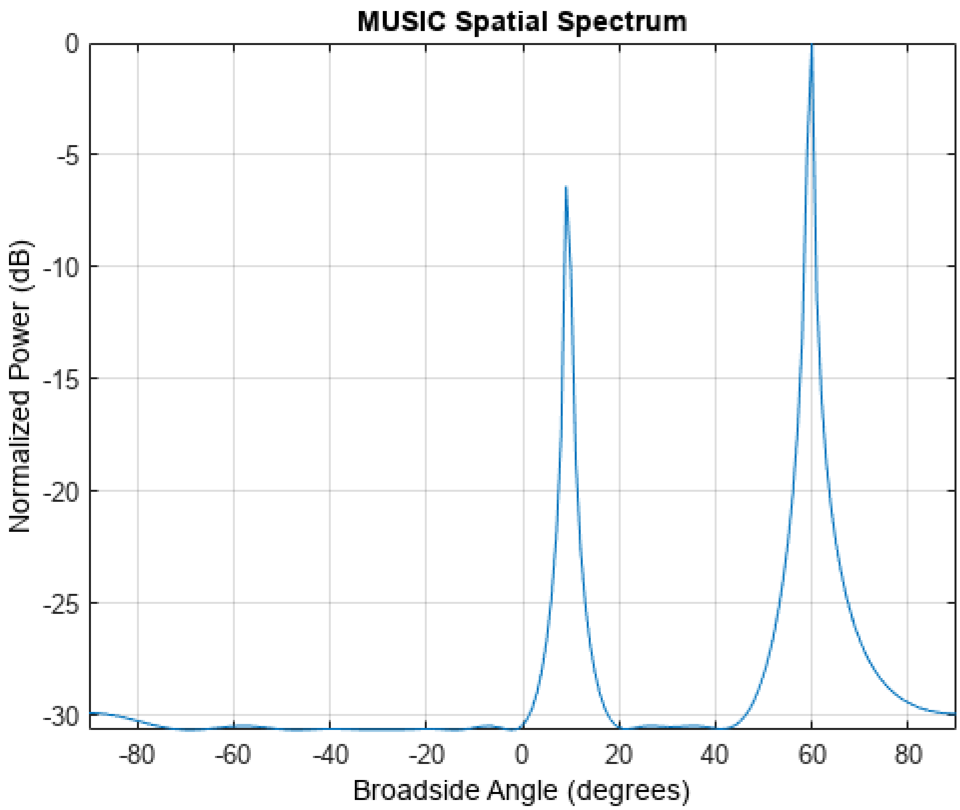
```
estimator = phased.MUSICEstimator('SensorArray',array,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignalsSource','Property',...
    'NumSignals',2);
```

```
[y,doas] = estimator(sig + noise);
doas = broadside2az(sort(doas),[20 -5])
```

```
doas = 1×2

    9.5829   60.3813
```

Plot the MUSIC spectrum.

```
plotSpectrum(estimator,'NormalizeResponse',true)
```



Compute DOA of Two Nearby Signals Using MUSIC

First, estimate the DOAs of two signals received by a standard 10-element ULA having an element spacing of one-half wavelength.Then, plot the spatial spectrum.

The antenna operating frequency is 150 MHz. The arrival directions of the two signals are separated by 2°. The direction of the first signal is 30° azimuth and 0° elevation. The direction of the second signal is 32° azimuth and 0° elevation. Estimate the number of signals using the Minimum Description Length (MDL) criterion.

Create the signals arriving at the ULA.

```
fs = 8000;
t = (0:1/fs:1).';
```

```
f1 = 300.0;
f2 = 600.0;
sig1 = cos(2*pi*t*f1);
sig2 = cos(2*pi*t*f2);
fc = 150.0e6;
c = physconst('LightSpeed');
lam = c/fc;
array = phased.ULA('NumElements',10,'ElementSpacing',0.5*lam);
sig = collectPlaneWave(array,[sig1 sig2],[30 0; 32 0]',fc);
noise = 0.1*(randn(size(sig)) + 1i*randn(size(sig)));
```

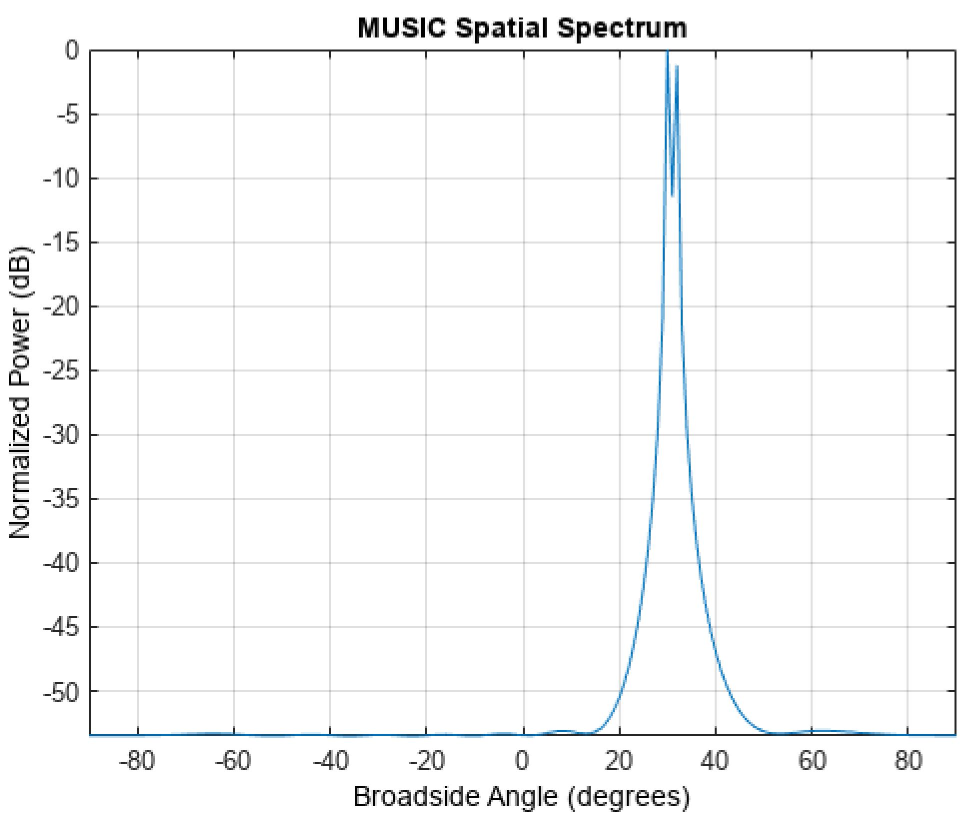
Estimate the DOAs.

```
estimator = phased.MUSICEstimator('SensorArray',array,...
    'OperatingFrequency',fc,'DOAOutputPort',true,...
    'NumSignalsSource','Auto','NumSignalsMethod','MDL');
[y,doas] = estimator(sig + noise);
doas = broadside2az(sort(doas),[0 0])
```

```
doas = 1×2

   30.0000   32.0000
```

Plot the MUSIC spectrum.

```
plotSpectrum(estimator,'NormalizeResponse',true)
```



Version History

Introduced in R2016b

phased.MUSICEstimator2D

Package: phased

Estimate 2D direction of arrival using narrowband MUSIC algorithm

Description

The phased.MUSICEstimator2D System object implements the narrowband multiple signal classification (MUSIC) algorithm for 2-D planar or 3-D arrays such as a uniform rectangular array (URA). MUSIC is a high-resolution direction-finding algorithm capable of resolving closely-spaced signal sources. The algorithm is based on the eigenspace decomposition of the sensor covariance matrix.

To estimate directions of arrival (DOA):

1. Define and set up a phased.MUSICEstimator2D System object. See "Construction" on page 1-1060.
2. Call the step method to estimate the DOAs according to the properties of phased.MUSICEstimator2D.

Note Instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

estimator = phased.MUSICEstimator2D creates a MUSIC DOA estimator System object, estimator.

estimator = phased.MUSICEstimator2D(Name,Value) creates a System object, estimator, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

SensorArray — Sensor array
phased.ULA array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object.

Example: phased.URA

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed').

Example: `3e8`

Data Types: `single` | `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `single` | `double`

`ForwardBackwardAveraging` — Enable forward-backward averaging
`false` (default) | `true`

Enable forward-backward averaging, specified as `false` or `true`. Set this property to `true` to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold.

Data Types: `logical`

`AzimuthScanAngles` — Azimuth scan angles
`[-90:90]` (default) | real-valued row vector

Azimuth scan angles, specified as a or real-valued row vector. Angle units are in degrees. The angle values must lie between –180° and 180°, inclusive, and be in ascending order.

Example: `[-30:20]`

Data Types: `single` | `double`

`ElevationScanAngles` — Elevation scan angles
`0` (default) | real-valued row vector

Elevation scan angles, specified as a real-valued row vector. Angle units are in degrees. The angle values must lie between –90° and 90°, inclusive, and be in ascending order.

Example: `[-70:75]`

Data Types: `single` | `double`

`DOAOutputPort` — Enable directions of arrival output
`false` (default) | `true`

Option to enable directions-of-arrival (DOA) output, specified as `false` or `true`. To obtain the DOA of signals, set this property to `true`. The DOAs are returned in the second output argument when the object is executed.

Data Types: `logical`

`NumSignalsSource` — Source of number of signals
`'Auto'` (default) | `'Property'`

Source of the number of arriving signals, specified as `'Auto'` or `'Property'`.

- `'Auto'` — The System object estimates the number of arriving signals using the method specified in the `NumSignalsMethod` property.

- 'Property' — Specify the number of arriving signals using the NumSignals property.

Data Types: char

NumSignalsMethod — Method used to estimate number of arriving signals
'AIC' (default) | 'MDL'

Method used to estimate the number of arriving signals, specified as 'AIC' or 'MDL'.

- 'AIC' — Akaike Information Criterion
- 'MDL' — Minimum Description Length criterion

Dependencies

To enable this property, set NumSignalsSource to 'Auto'.

Data Types: char

NumSignals — Number of arriving signals
1 (default) | positive integer

Number of arriving signals for DOA estimation, specified as a positive integer.

Example: 3

Dependencies

To enable this property, set NumSignalsSource to 'Property'.

Data Types: single | double

Methods

plotSpectrum	Plot 2-D MUSIC spectrum
reset	Reset states of System object
step	Estimate direction of arrival using 2-D MUSIC

Common to All System Objects	
release	Allow System object property value changes

Examples

Estimate DOAs of Two Signals

Assume that two sinusoidal waves of frequencies 450 Hz and 600 Hz strike a URA from two different directions. Signals arrive from -37° azimuth, 0° elevation and 17° azimuth, 20° elevation. Use 2-D MUSIC to estimate the directions of arrival of the two signals. The array operating frequency is 150 MHz and the signal sampling frequency is 8 kHz.

```
f1 = 450.0;
f2 = 600.0;
doa1 = [-37;0];
doa2 = [17;20];
```

```
fc = 150e6;
c = physconst('LightSpeed');
lam = c/fc;
fs = 8000;
```

Create the URA with default isotropic elements. Set the frequency response range of the elements.

```
array = phased.URA('Size',[11 11],'ElementSpacing',[lam/2 lam/2]);
array.Element.FrequencyRange = [50.0e6 500.0e6];
```

Create the two signals and add random noise.

```
t = (0:1/fs:1).';
x1 = cos(2*pi*t*f1);
x2 = cos(2*pi*t*f2);
x = collectPlaneWave(array,[x1 x2],[doa1,doa2],fc);
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
```

Create and execute the 2-D MUSIC estimator to find the directions of arrival.

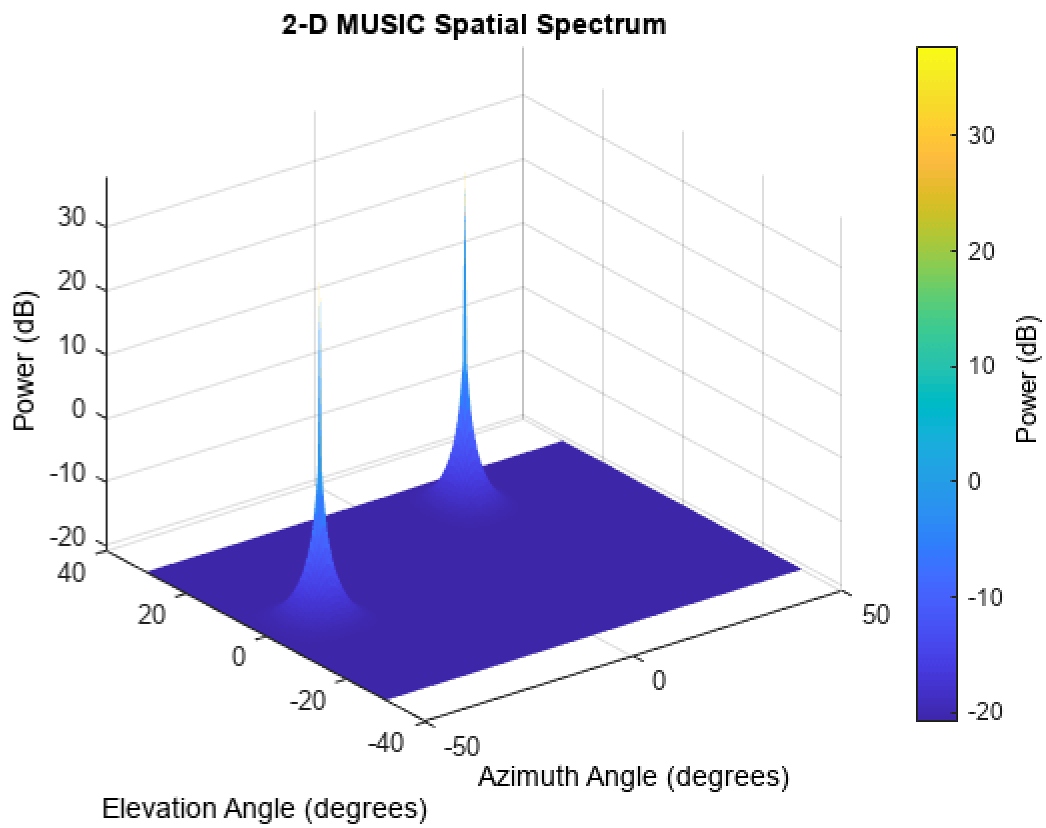
```
estimator = phased.MUSICEstimator2D('SensorArray',array,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property',...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:.5:50,...
    'ElevationScanAngles',-30:.5:30);
[~,doas] = estimator(x + noise)
```

```
doas = 2×2

   -37    17
     0    20
```

The estimated DOAs exactly match the true DOAs.

Plot the 2-D spatial spectrum.

```
plotSpectrum(estimator);
```



Estimate DOAs of Two Signals at Disk Array

Assume that two sinusoidal waves of frequencies 1.6 kHz and 1.8 kHz strike a disk array from two different directions. The spacing between elements of the disk is 1/2 wavelength. Signals arrive from -31° azimuth, -11° elevation and 35° azimuth, 55° elevation. Use 2-D MUSIC to estimate the directions of arrival of the two signals. The array operating frequency is 300 MHz and the signal sampling frequency is 8 kHz.

```
f1 = 1.6e3;
f2 = 1.8e3;
doa1 = [-31;-11];
doa2 = [35;55];
fc = 300e6;
c = physconst('LightSpeed');
lam = c/fc;
fs = 8.0e3;
```

Create a conformal array with default isotropic elements. First, create a URA to get the element positions.

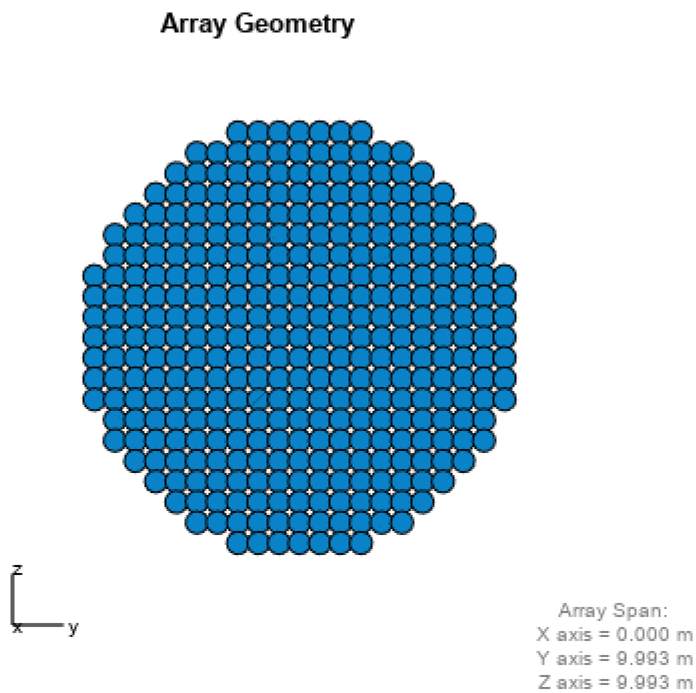
```
uraarray = phased.URA('Size',[21 21],'ElementSpacing',[lam/2 lam/2]);
pos = getElementPosition(uraarray);
```

Extract a subset of these to form an inscribed disk.

```
radius = 10.5*lam/2;
pos(:,sum(pos.^2) > radius^2) = [];
```

Then, create the conformal array using these positions.

```
confarray = phased.ConformalArray('ElementPosition',pos);
viewArray(confarray)
```



Set the frequency response range of the elements.

```
confarray.Element.FrequencyRange = [50.0e6 600.0e6];
```

Create the two signals and add random noise.

```
t = (0:1/fs:1.5).';
x1 = cos(2*pi*t*f1);
x2 = cos(2*pi*t*f2);
x = collectPlaneWave(confarray,[x1 x2],[doa1,doa2],fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
```

Create and execute the 2-D MUSIC estimator to find the directions of arrival.

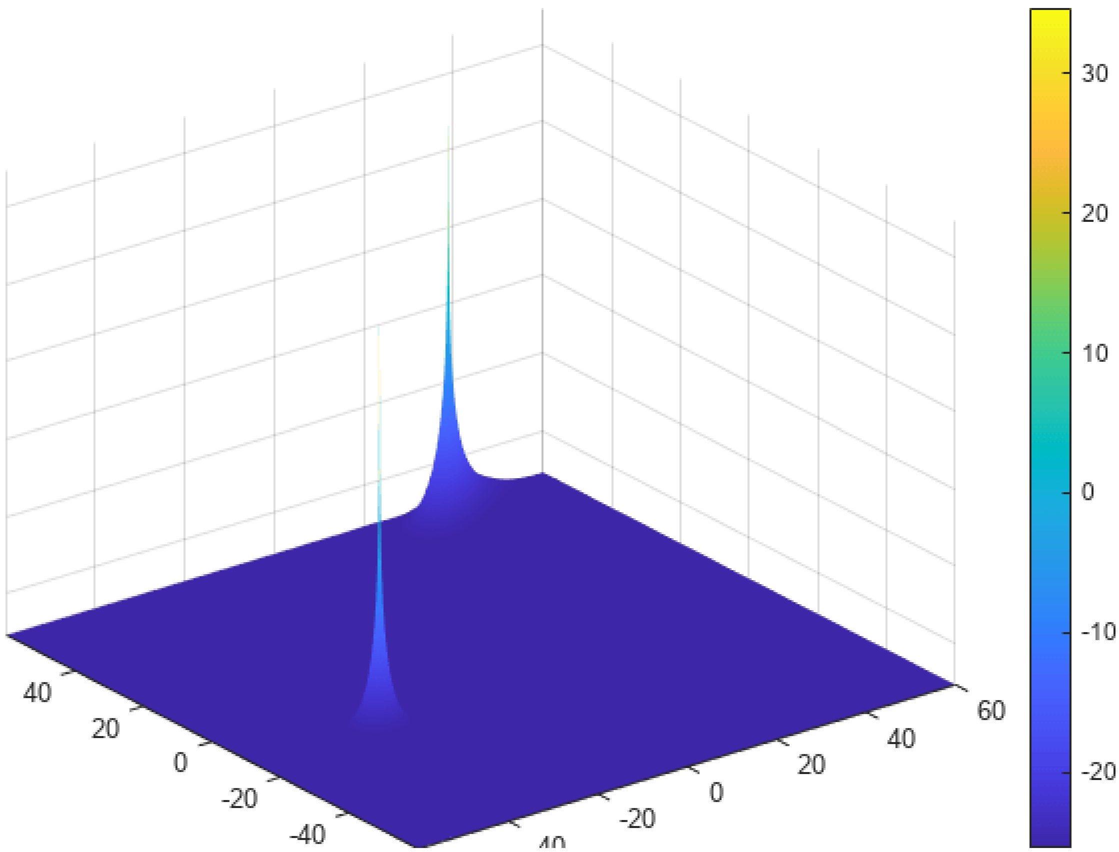
```
estimator = phased.MUSICEstimator2D('SensorArray',confarray,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property',...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-60:.1:60,...
    'ElevationScanAngles',-60:.1:60);
[~,doas] = estimator(x + noise)
```

```
doas = 2×2

    35   -31
    55   -11
```

The estimated DOAs exactly match the true DOAs.

Plot the 2-D spatial spectrum.

```
plotSpectrum(estimator);
```



Algorithms

MUSIC Algorithm

MUSIC stands for *MUltiple SIgnal Classification*. MUSIC is a high-resolution direction-finding algorithm that estimates directions of arrival (DOA) of signals at an array from the covariance matrix of array sensor data. MUSIC belongs to the subspace-decomposition family of direction-finding algorithms. Unlike conventional beamforming, MUSIC can resolve closely-spaced signal sources.

Based on eigenspace decomposition of the sensor covariance matrix, MUSIC divides the observation space into orthogonal signal and noise subspaces. Eigenvectors corresponding to the largest eigenvalues span the signal subspace. Eigenvectors corresponding to the smaller eigenvalues span the noise subspace. Because arrival (or steering) vectors lie in the signal subspace, they are orthogonal to the noise subspace. The arrival vectors depend on the direction of arrival of a signals. For a 2-D or 3-D array. the directions are determined by the azimuth and elevation of the sources. By searching over a grid of arrival angles, the algorithm finds those arrival vectors whose projection into the noise subspace is zero or at least very small.

MUSIC requires that the number of source signals is known. The algorithm degrades if the number of specified sources does not match the actual number of sources. Generally, you must provide an estimate of the number of sources or use one of the built-in source number estimation methods. For a description of the methods used to estimate the number of sources, see the `aictest` or `mdltest` functions.

In place of the true sensor covariance matrix, the algorithm computes the sample covariance matrix from the sensor data. MUSIC applies to noncoherent signals but can be extended to coherent signals using forward-backward averaging techniques. For a high-level description of the algorithm, see "MUSIC Super-Resolution DOA Estimation".

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2016b

References

[1] Van Trees, H. L., *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

Functions

`aictest` | `mdltest` | `musicdoa` | `rootmusicdoa`

Objects

`phased.MUSICEstimator` | `phased.RootMUSICEstimator`

Topics

"MUSIC Super-Resolution DOA Estimation"
"Direction of Arrival Estimation with Beamscan, MVDR, and MUSIC"
"High Resolution Direction of Arrival Estimation"
"Spherical Coordinates"

plotSpectrum

System object: `phased.MUSICEstimator2D`
Package: `phased`

Plot 2-D MUSIC spectrum

Syntax

```
plotSpectrum(estimator)
output_args = plotSpectrum(estimator,Name,Value)
lh = plotSpectrum( ___ )
```

Description

`plotSpectrum(estimator)` plots the 2-D MUSIC spatial spectrum computed by the most recent `step` method execution for the `phased.MUSICEstimator2D`, `estimator`.

`output_args = plotSpectrum(estimator,Name,Value)` plots the 2-D MUSIC spatial spectrum with additional options specified by one or more `Name,Value` pair arguments.

`lh = plotSpectrum( ___ )` returns the line handle to the figure.

Input Arguments

`estimator` — 2-D MUSIC estimator
`phased.MUSICEstimator2D` System object

2-D MUSIC estimator, specified as a `phased.MUSICEstimator2D` System object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Unit` — Units used for plotting
`'db'` (default) | `'mag'` | `'pow'`

Units used for plotting, specified as the comma-separated pair consisting of `'Unit'` and `'db'`, `'mag'`, or `'pow'`.

Data Types: `char`

`NormalizeResponse` — Plot normalized spectrum
`false` (default) | `true`

Plot a normalized spectrum, specified as the comma-separated pair consisting of `'NormalizedResponse'` and `false` or `true`. Normalization sets the magnitude of the largest spectrum value to one.

Example: true

Data Types: char

Title — Title of plot
'2D MUSIC Spatial Spectrum' (default) | character vector

Title of plot, specified as a comma-separated pair consisting of 'Title' and a character vector.

Example: true

Data Types: char

Output Arguments

lh — Line handle of plot
line handle

Line handle of plot.

Examples

Estimate DOAs of Two Signals

Assume that two sinusoidal waves of frequencies 450 Hz and 600 Hz strike a URA from two different directions. Signals arrive from -37° azimuth, 0° elevation and 17° azimuth, 20° elevation. Use 2-D MUSIC to estimate the directions of arrival of the two signals. The array operating frequency is 150 MHz and the signal sampling frequency is 8 kHz.

```
f1 = 450.0;
f2 = 600.0;
doa1 = [-37;0];
doa2 = [17;20];
fc = 150e6;
c = physconst('LightSpeed');
lam = c/fc;
fs = 8000;
```

Create the URA with default isotropic elements. Set the frequency response range of the elements.

```
array = phased.URA('Size',[11 11],'ElementSpacing',[lam/2 lam/2]);
array.Element.FrequencyRange = [50.0e6 500.0e6];
```

Create the two signals and add random noise.

```
t = (0:1/fs:1).';
x1 = cos(2*pi*t*f1);
x2 = cos(2*pi*t*f2);
x = collectPlaneWave(array,[x1 x2],[doa1,doa2],fc);
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
```

Create and execute the 2-D MUSIC estimator to find the directions of arrival.

```
estimator = phased.MUSICEstimator2D('SensorArray',array,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property',...
```

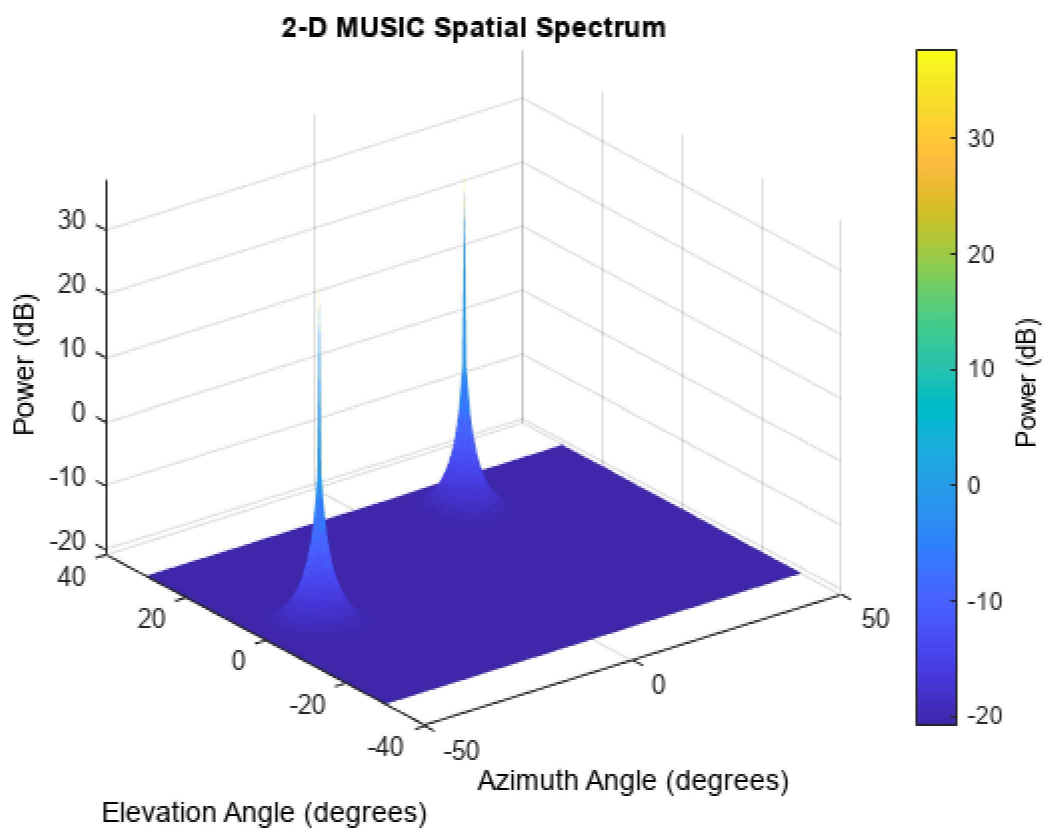
```
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:.5:50,...
    'ElevationScanAngles',-30:.5:30);
[~,doas] = estimator(x + noise)
```

```
doas = 2×2

   -37    17
     0    20
```

The estimated DOAs exactly match the true DOAs.

Plot the 2-D spatial spectrum.

```
plotSpectrum(estimator);
```



Version History

Introduced in R2016b

reset

System object: `phased.MUSICEstimator2D`
Package: `phased`

Reset states of System object

Syntax

`reset(estimator)`

Description

`reset(estimator)` resets the internal state of the `phased.MUSICEstimator2D` System object, `estimator`.

Input Arguments

`estimator` — 2-D MUSIC estimator
`phased.MUSICEstimator2D` System object

2-D MUSIC estimator, specified as a `phased.MUSICEstimator2D` System object.

Version History

Introduced in R2016b

step

System object: phased.MUSICEstimator2D
Package: phased

Estimate direction of arrival using 2-D MUSIC

Syntax

```
spectrum = step(estimator,X)
[spectrum,doa] = step(estimator,X)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`spectrum = step(estimator,X)` returns the 2-D MUSIC spectrum of a signal specified in `X`.

`[spectrum,doa] = step(estimator,X)` also returns the signal directions of arrival angles, `doa`. To use this syntax, set the `DOAOutputPort` property to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`estimator` — 2-D MUSIC estimator

`phased.MUSICEstimator2D` System object

2-D MUSIC estimator, specified as a `phased.MUSICEstimator2D` System object.

`X` — Received signal

M-by-*N* complex-valued matrix

Received signal, specified as an *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values (snapshots) contained in the signal and *N* is the number of sensor elements in the array.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[[0;1;2;3;4;3;2;1;0],[1;2;3;4;3;2;1;0;0]]`

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

`spectrum` — 2-D MUSIC spatial spectrum
nonnegative, real-valued *K*-length column vector

2-D MUSIC spatial spectrum, returned as a nonnegative, real-valued *K*-length column vector representing the magnitude of the estimated MUSIC spatial spectrum. Each entry corresponds to an angle specified by the `AzimuthScanAngles` and `ElevationScanAngles` properties.

`doa` — Directions of arrival
real-valued 2-by-*L* matrix

Directions of arrival of the signals, returned as a real-valued 2-by-*L* matrix. The direction of arrival angle is defined by the azimuth and elevation angles of the source with respect to the array local coordinate system. The first row of the matrix contains the azimuth angles and the second row contains the elevation angles. Angle units are in degrees. *L* is the number of signals specified by the `NumSignals` property or derived using the method specified by the `NumSignalsMethod` property.

Dependencies

To enable this output argument, set the `DOAOutputPort` property to `true`.

Examples

Estimate DOAs of Two Signals

Assume that two sinusoidal waves of frequencies 450 Hz and 600 Hz strike a URA from two different directions. Signals arrive from -37° azimuth, 0° elevation and 17° azimuth, 20° elevation. Use 2-D MUSIC to estimate the directions of arrival of the two signals. The array operating frequency is 150 MHz and the signal sampling frequency is 8 kHz.

```
f1 = 450.0;
f2 = 600.0;
doa1 = [-37;0];
doa2 = [17;20];
fc = 150e6;
c = physconst('LightSpeed');
lam = c/fc;
fs = 8000;
```

Create the URA with default isotropic elements. Set the frequency response range of the elements.

```
array = phased.URA('Size',[11 11],'ElementSpacing',[lam/2 lam/2]);
array.Element.FrequencyRange = [50.0e6 500.0e6];
```

Create the two signals and add random noise.

```
t = (0:1/fs:1).';
x1 = cos(2*pi*t*f1);
x2 = cos(2*pi*t*f2);
x = collectPlaneWave(array,[x1 x2],[doa1,doa2],fc);
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
```

Create and execute the 2-D MUSIC estimator to find the directions of arrival.

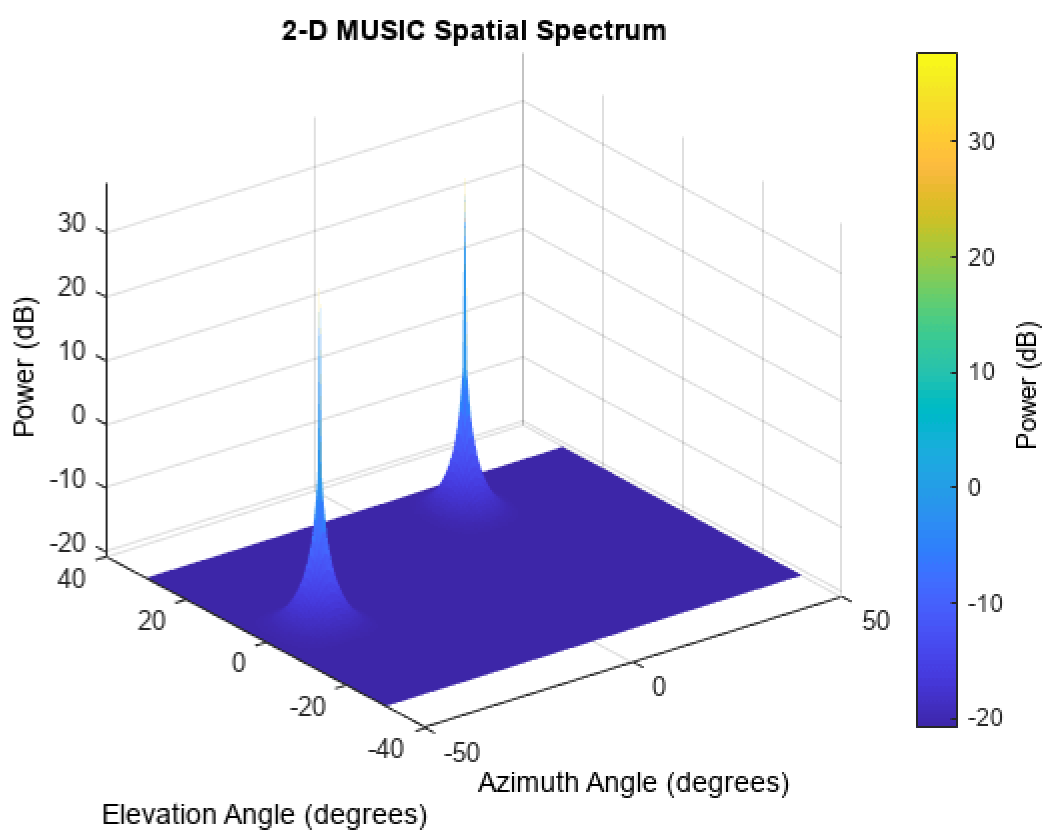
```
estimator = phased.MUSICEstimator2D('SensorArray',array,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property',...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:.5:50,...
    'ElevationScanAngles',-30:.5:30);
[~,doas] = estimator(x + noise)
```

```
doas = 2×2

   -37    17
     0    20
```

The estimated DOAs exactly match the true DOAs.

Plot the 2-D spatial spectrum.

```
plotSpectrum(estimator);
```



Estimate DOAs of Two Signals at Disk Array

Assume that two sinusoidal waves of frequencies 1.6 kHz and 1.8 kHz strike a disk array from two different directions. The spacing between elements of the disk is 1/2 wavelength. Signals arrive from

-31° azimuth, -11° elevation and 35° azimuth, 55° elevation. Use 2-D MUSIC to estimate the directions of arrival of the two signals. The array operating frequency is 300 MHz and the signal sampling frequency is 8 kHz.

```
f1 = 1.6e3;
f2 = 1.8e3;
doa1 = [-31;-11];
doa2 = [35;55];
fc = 300e6;
c = physconst('LightSpeed');
lam = c/fc;
fs = 8.0e3;
```

Create a conformal array with default isotropic elements. First, create a URA to get the element positions.

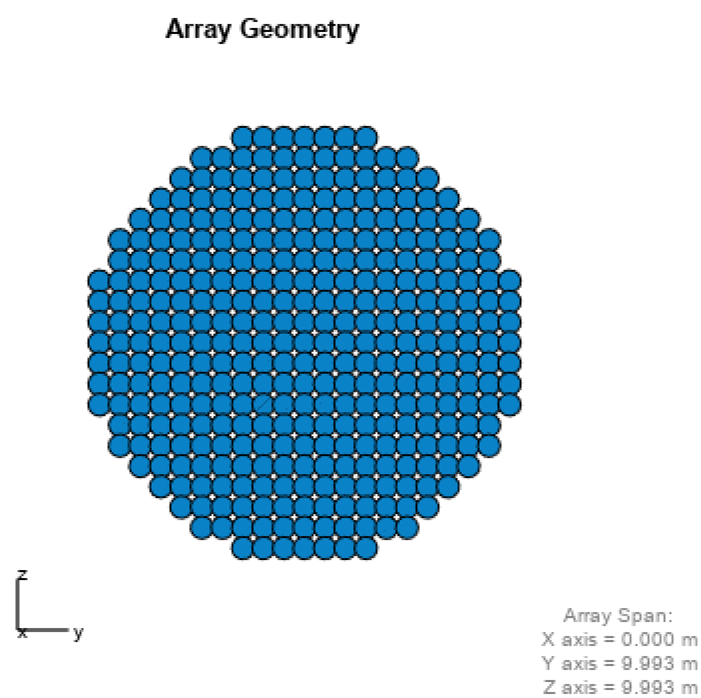
```
uraarray = phased.URA('Size',[21 21],'ElementSpacing',[lam/2 lam/2]);
pos = getElementPosition(uraarray);
```

Extract a subset of these to form an inscribed disk.

```
radius = 10.5*lam/2;
pos(:,sum(pos.^2) > radius^2) = [];
```

Then, create the conformal array using these positions.

```
confarray = phased.ConformalArray('ElementPosition',pos);
viewArray(confarray)
```



Set the frequency response range of the elements.

```
confarray.Element.FrequencyRange = [50.0e6 600.0e6];
```

Create the two signals and add random noise.

```
t = (0:1/fs:1.5).';
x1 = cos(2*pi*t*f1);
x2 = cos(2*pi*t*f2);
x = collectPlaneWave(confarray,[x1 x2],[doa1,doa2],fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
```

Create and execute the 2-D MUSIC estimator to find the directions of arrival.

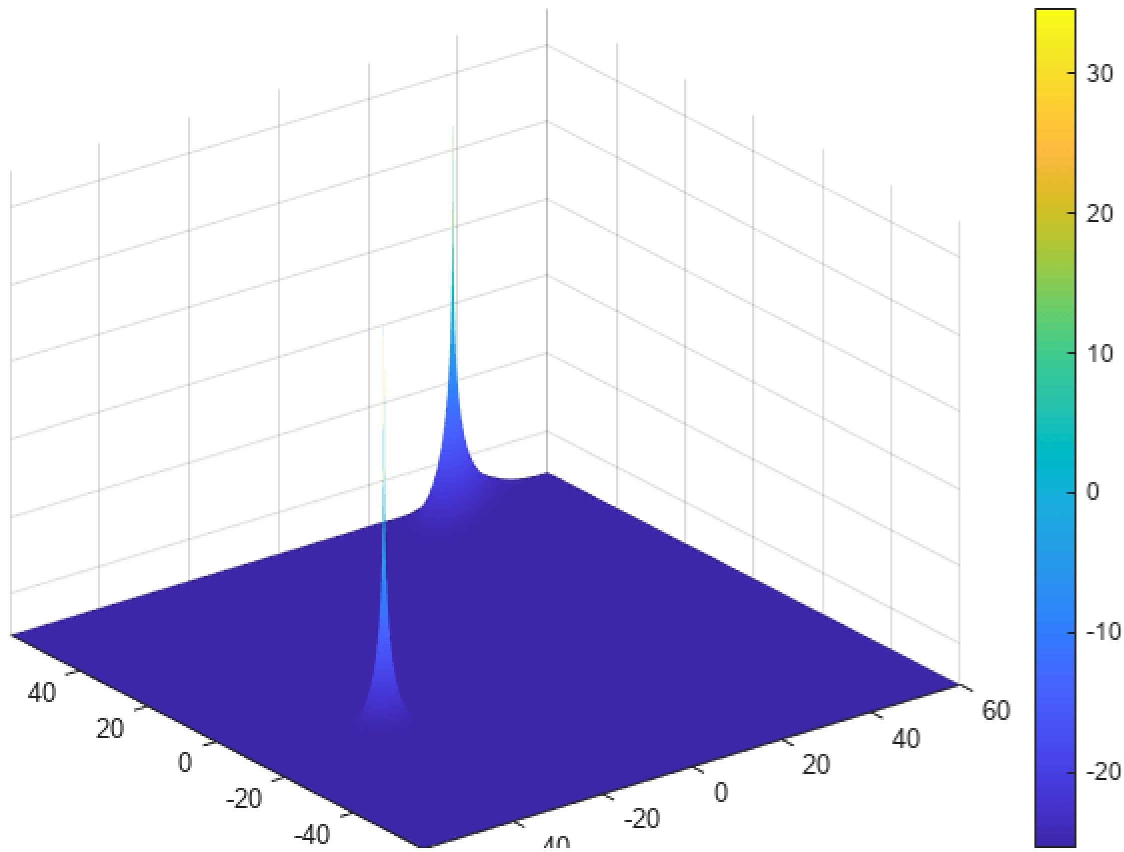
```
estimator = phased.MUSICEstimator2D('SensorArray',confarray,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property',...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-60:.1:60,...
    'ElevationScanAngles',-60:.1:60);
[~,doas] = estimator(x + noise)
```

```
doas = 2×2

    35   -31
    55   -11
```

The estimated DOAs exactly match the true DOAs.

Plot the 2-D spatial spectrum.

```
plotSpectrum(estimator);
```



Version History

Introduced in R2016b

phased.NRAntennaElement

Package: phased

5G antenna element described in 3GPP TR 38.901 specification

Description

The NRAntennaElement System object models an antenna designed to meet the 3GPP TR 38.901 standard [1].

To compute the response of the antenna element for specified directions:

1. Create the phased.NRAntennaElement object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
antenna = phased.NRAntennaElement
antenna = phased.NRAntennaElement(Name,Value)
```

Description

antenna = phased.NRAntennaElement creates an NR antenna System object, antenna, that follows the standard specified in 3GPP TR 38.901 [1].

antenna = phased.NRAntennaElement(Name,Value) creates an NR antenna element object, antenna, and sets each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

FrequencyRange — Operating frequency range
[0 1e20] (default) | nonnegative, real-valued 1-by-2 row vector

Operating frequency range of the antenna, specified as a nonnegative, real-valued, 1-by-2 row vector in the form [LowerBound HigherBound]. The antenna element has no response outside the specified frequency range. Units are in Hz.

Data Types: `double`

`PolarizationAngle` — Polarization slant angle
`0.0` (default) | real scalar

Polarization slant angle of the antenna, specified as a scalar. The polarization slant angle is defined in section 7.3.2 of the 3GPP TR 38.901 Release 14 [1]. Units are in degrees.

Example: `45.0`

Data Types: `double`

`PolarizationModel` — Polarization model
`2` (default) | `1`

Polarization model, specified as either `1` or `2`. The polarization models are defined in section 7.3.2 of the 3GPP TR 38.901 Release 14 [1].

Example: `1`

Data Types: `double`

`Beamwidth` — Beamwidth of antenna pattern
`[65 65]` (default) | scalar | 1-by-2 real-valued vector

Beamwidth of the antenna pattern, specified as either a scalar or a 1-by-2 real-valued vector. When the specified value is a 1-by-2 vector, it has the form of `[AzimuthBeamwidth ElevationBeamwidth]`. If the specified value is a scalar, the azimuth and elevation beamwidths are equal. Units are in degrees.

Example: `40`

Data Types: `double`

`SidelobeLevel` — Attenuation of maximum sidelobe level
`[30 30]` (default) | positive scalar | 1-by-2 real-valued vector of positive numbers

Attenuation of maximum sidelobe level of the antenna pattern, specified as either a positive scalar or a 1-by-2 real-valued vector of positive numbers. When the specified value is a 1-by-2 vector, it has the form of `[AzimuthSidelobe ElevationSidelobe]`. If the specified value is a scalar, the azimuth and elevation sidelobe levels are equal. Units are in dB.

Example: `24`

Data Types: `double`

`MaximumAttenuation` — Maximum pattern attenuation
`30` (default) | positive scalar

Maximum attenuation to the main lobe in the antenna pattern specified as a positive scalar. This value should be no less than the values specified in the `SidelobeLevel` property. Units are in dB.

Example: `28`

Data Types: `double`

`MaximumGain` — Maximum gain in the main lobe
`8` (default) | positive scalar

Maximum gain of the main lobe in the antenna pattern specified as a positive scalar. Units are in dB.

Example: 5

Data Types: `double`

Usage

Syntax

`RESP = antenna(FREQ,ANG)`

Description

`RESP = antenna(FREQ,ANG)` returns the antenna voltage response `RESP` at the operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`FREQ` — Operating frequency of antenna element
nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

`FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the response is returned as `–Inf`. Element objects use the `FrequencyRange` property, except for `phased.CustomAntennaElement`, which uses the `FrequencyVector` property.

Example: `[1e8 2e6]`

Data Types: `double`

`ANG` — Azimuth and elevation angles of response directions
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If `ANG` is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of “Azimuth and Elevation Angles”.

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

RESP — Voltage response of antenna
complex-valued *M*-by-*L* matrix

Voltage response of the antenna element, returned as a complex-valued *M*-by-*L* matrix. In this matrix, *M* represents the number of angles specified in `ANG` and *L* represents the number of frequencies specified in `FREQ`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

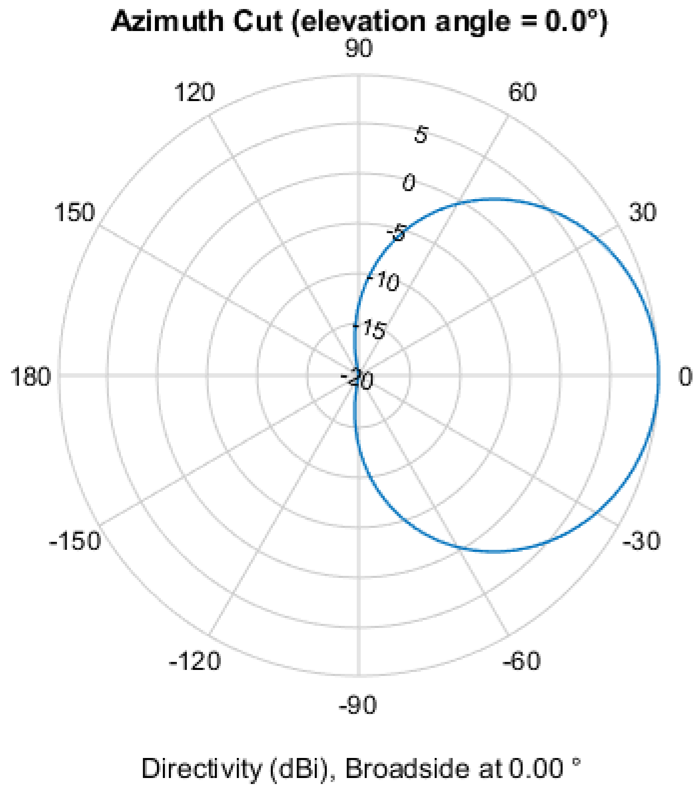
step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Create NR Antenna Element

Construct an antenna based on the 3GPP 38.901 standard and plot its elevation response at 6 MHz.

```
antenna = phased.NRAntennaElement;
fc = 6e9;
pattern(antenna,fc,-180:180,0,'CoordinateSystem','polar');
```



Find the response of the antenna at the boresight.

```
ang = [0;0];
resp = antenna(fc,ang)
```

```
resp = struct with fields:
    H: 0
    V: -2.5119
```

Construct NR Antenna with Polarization Model

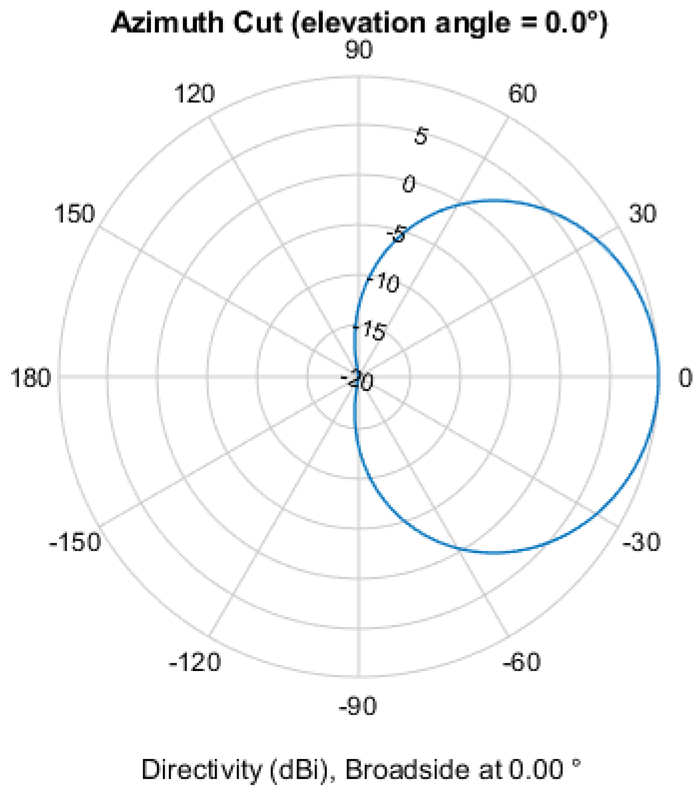
Construct an NR antenna based on the 3GPP 38.901 standard with its Polarization Model set to "1". Then, find its response at boresight. Finally, plot its antenna response as a function of azimuth angle at 6 GHz.

```
element = phased.NRAntennaElement('PolarizationModel',1);
fc = 6e9;
ang = [0;0];
resp = element(fc,ang)
```

```
resp = struct with fields:
    H: 0
    V: -2.5119
```

Plot the antenna pattern at 0 degrees elevation for all azimuth angles.

```
pattern(element,fc,-180:180,0,'CoordinateSystem','polar')
```



Construct NR Antenna with Specified Beamwidth

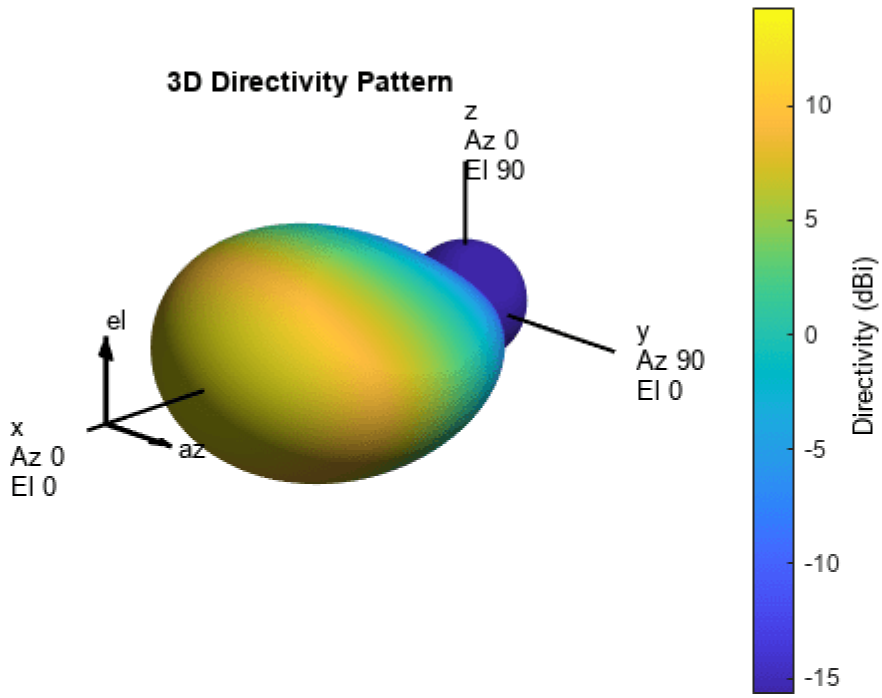
Construct an NR antenna based on the 3GPP 38.901 standard. Set the antenna beamwidth to 45 degrees in azimuth and 30 degrees in elevation. Find the antenna response at boresight. Then, plot the antenna response as a function of azimuth and elevation at 6 GHz.

```
element = phased.NRAntennaElement('Beamwidth',[45,30]);
fc = 6.0e9;
ang = [0;0];
resp = element(fc,ang)
```

```
resp = struct with fields:
    H: 0
    V: -2.5119
```

Plot the 3D antenna pattern for all azimuth angles and elevation angles.

```
pattern(element,fc,-180:180,-90:90,'CoordinateSystem','polar')
```



Version History

Introduced in R2021a

References

[1] *5G: Study on channel model for frequencies from 0.5 to 100 GHz,* 3GPP TR38.901 Version 14.0.0 Release 14.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

“System Objects in MATLAB Code Generation” (MATLAB Coder)

See Also

`phased.NRRectangularPanelArray`

phased.NRRectangularPanelArray

Package: `phased`

5G antenna array described in 3GPP TR 38.901 specification

Description

The `phased.NRRectangularPanelArray` System object creates a rectangular antenna array designed to meet the 3GPP TR 38.901 standard. This object models an antenna pattern generated by multiple panels in a rectangular layout. Each panel is a heterogeneous array consisting of co-located antenna elements. The default set of antenna elements is a pair of `phased.NRAntennaElement` antenna elements having +45° and -45° as the polarization slant angles. The default configuration of each panel is a 2-by-2 antenna configuration. You can also create an array using other types of elements. Elements and panels lie in the *yz*-plane.

To compute the response of the antenna element for specified directions:

1. Create the `phased.NRRectangularPanelArray` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
array = phased.NRRectangularPanelArray
array = phased.NRRectangularPanelArray(Name,Value)
```

Description

`array = phased.NRRectangularPanelArray` creates an NR antenna panel array System object, `array` that follows the specification described in the 3GPP TR 38.901.

`array = phased.NRRectangularPanelArray(Name,Value)` creates an NR rectangular panel array object, `array`, with each specified property set to the specified value. You can specify additional name-value arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

ElementSet — Antenna elements
1-by-2 cell array (default) | 1-by-1 cell array

Antenna elements, specified as a 1-by-2 cell array or 1-by-1 cell array. The default value is a 1-by-2 cell array containing two phased.NRAntennaElement antenna elements with a selected polarization model of 2 – one with a –45° polarization angle and the other with a +45° polarization angle.

Size — Sizes of element grid and panel grid
[2 2 1 1] (default) | 1-by-4 vector of positive integer values

Sizes of element grid and panel grid, specified as a 1-by-4 vector of positive integer values. The first two entries represent the number of elements in a panel and the last two entries represent the number of panels in the array.. The default value is one panel grid with a panel size of 2-by-2 elements.

Example: [2 2 2 2]

Data Types: double

Spacing — Spacing between elements and between panels
[0.5 0.5 1.0 1.0] (default) | 1-by-4 vector of real positive values

Spacing between elements and between panels, specified as a 1-by-4 positive vector. The first two entries represent the spacing between elements within the panel and the remaining values in the represent the spacing between panels in the array. Units are in meters.

Example: [0.75 0.75 1.0 1.0]

Data Types: double

Taper — Array element tapers
1 (default) | complex scalar | length-*N* complex-valued vector

Array element tapers, specified as a complex scalar or a length-*N* complex-valued vector of weights applied to each element in the sensor array. *N* is the number of elements in the array. If Taper is a scalar, identical weights are applied to each element. If Taper is a vector, each weight is applied to the corresponding array element.

Example: 1

Data Types: double
Complex Number Support: Yes

Usage

Syntax

```
RESP = array(FREQ,ANG)
```

Description

RESP = array(FREQ,ANG) returns the array voltage response, RESP, at the operating frequencies specified in FREQ and in directions specified in ANG.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`FREQ` — Operating frequency of antenna element

nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

`FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the response is returned as `–Inf`. Element objects use the `FrequencyRange` property, except for `phased.CustomAntennaElement`, which uses the `FrequencyVector` property.

Example: `[1e8 2e6]`

Data Types: `double`

`ANG` — Azimuth and elevation angles of response directions

real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If `ANG` is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth;elevation]`.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of "Azimuth and Elevation Angles".

Example: `[110 125; 15 10]`

Data Types: `double`

Output Arguments

`RESP` — Voltage response of array

complex-valued *M*-by-*L* matrix

Voltage response of the array, returned as a complex-valued *M*-by-*L* matrix. In this matrix, *M* represents the number of angles specified in `ANG` and *L* represents the number of frequencies specified in `FREQ`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Array System Objects

beamwidth	Compute and display beamwidth of an array
collectPlaneWave	Simulate received plane waves at array
directivity	Compute array directivity
getElementNormal	Normal vectors for array elements
getElementPosition	Positions of array elements
getNumElements	Number of elements in an array
getTaper	Array element tapers
isPolarizationCapable	Array polarization capability
pattern	Plot array directivity and patterns
patternAzimuth	Plot array directivity or pattern versus azimuth
patternElevation	Plot array directivity or pattern versus elevation
perturbations	Perturbations defined on array
perturbedArray	Apply perturbations to phased array
perturbedPattern	Display pattern of perturbed array
viewArray	View array geometry

Common to All System Objects

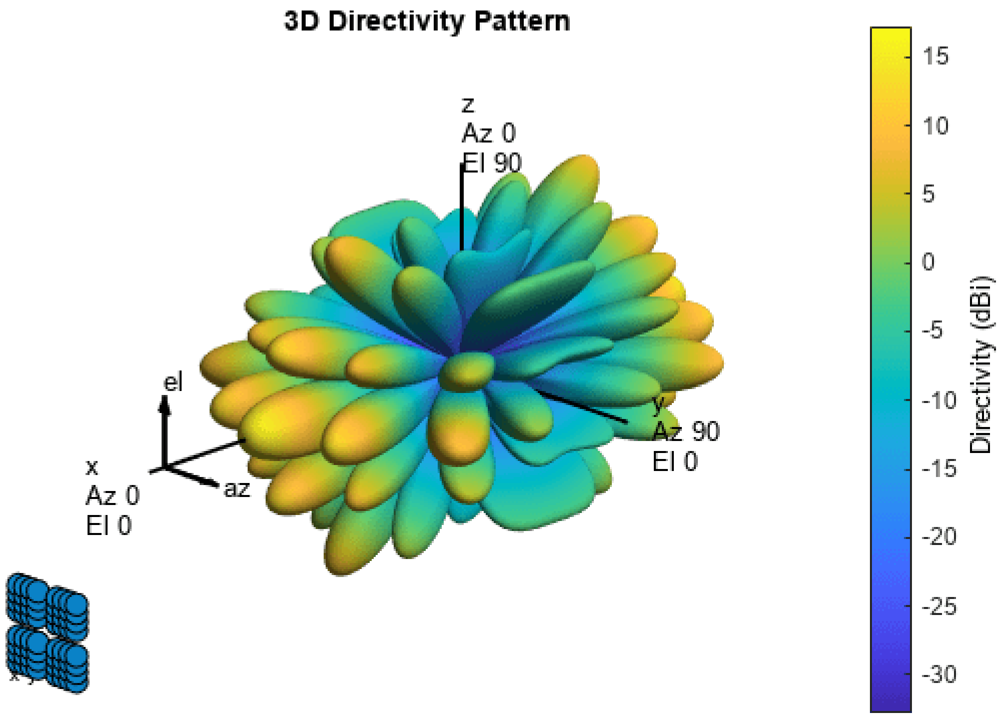
step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Plot Response of NR Rectangular Panel Array

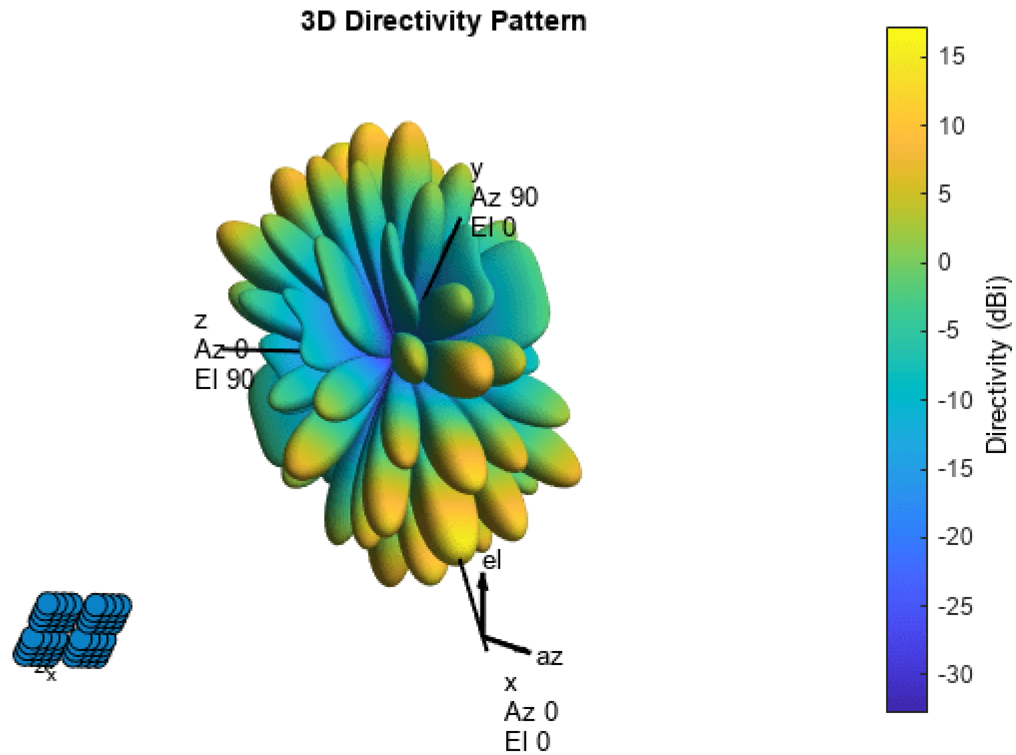
Construct a 5G antenna array where the grid is 2-by-2 and each panel is a 4-by-4 array. Each antenna element consists of two short-dipole antennas with different dipole axis directions. The antenna elements are spaced 1/2 wavelength apart and the panels are spaced 3 wavelengths apart. Plot the response pattern of the array assuming an operating frequency of 6 GHz.

```
c = physconst('LightSpeed');
fc = 6e9;
lambda = c/fc;
antenna1 = phased.ShortDipoleAntennaElement('AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('AxisDirection','X');
array = phased.NRRectangularPanelArray('ElementSet', ...
        {antenna1, antenna2},'Size',[4, 4, 2, 2],'Spacing', ...
        [0.5*lambda, 0.5*lambda,3*lambda, 3*lambda]);
pattern(array,fc,'ShowArray',true)
```



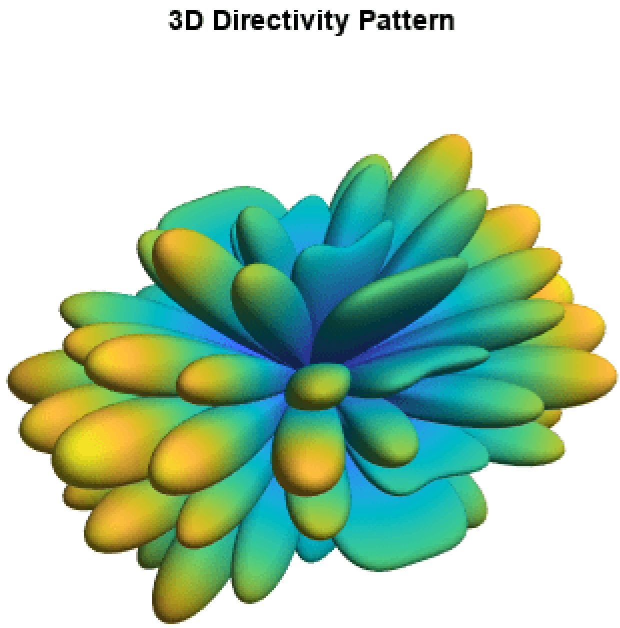
Use the `Orientation` property of `pattern` to change the orientation 80˚ along the *x*-axis, 30˚ along the *y*-axis and 60˚ along the *z*-axis.

```
pattern(array,fc,'Orientation',[80;30;60],'ShowArray',true)
```



Disable the display of local coordinates and the colorbar.

```
pattern(array,fc,'ShowLocalCoordinate',false,'ShowColorBar',false)
```



Get Response of NR Rectangular Array

Construct a 5G antenna array where the grid is 2-by-2 and each panel is a 3-by-2 array. The antenna elements are two **phased.NRAntennaElement** objects having polarization angles of +45 and −45 degrees. Find the response of the array at boresight, assuming an operating frequency of 6 GHz. The elements are spaced 1/2-wavelength apart and the panels are 3-wavelengths apart.

```
c = physconst('LightSpeed');
fc = 6e9;2
```

```
ans = 2
```

```
lambda = c/fc;
array = phased.NRRectangularPanelArray('Size',[3, 2, 2, 2], ...
        'Spacing',[0.5*lambda,0.5*lambda,3*lambda,3*lambda]);
resp = array(fc,[0;0])
```

```
resp = struct with fields:
    H: [48x1 double]
    V: [48x1 double]
```

Version History

Introduced in R2021a

References

[1] *5G: Study on channel model for frequencies from 0.5 to 100 GHz*, 3GPP TR38.901 Version 14.0.0 Release 14.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

“System Objects in MATLAB Code Generation” (MATLAB Coder)

See Also

`phased.NRAntennaElement`

phased.OmnidirectionalMicrophoneElement

Package: `phased`

Omnidirectional microphone element

Description

The `phased.OmnidirectionalMicrophoneElement` System object models a microphone element with an omnidirectional response pattern.

To compute the response of the microphone element for specified directions:

1. Create the `phased.OmnidirectionalMicrophoneElement` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
microphone = phased.OmnidirectionalMicrophoneElement
microphone = phased.OmnidirectionalMicrophoneElement(Name=Value)
```

Description

`microphone = phased.OmnidirectionalMicrophoneElement` creates an omnidirectional microphone System object, `microphone`, with default object properties.

`microphone = phased.OmnidirectionalMicrophoneElement(Name=Value)` creates an omnidirectional microphone object, `microphone`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1=Value1`,...,`NameN=ValueN`).

Example: `microphone = phased.OmnidirectionalMicrophoneElement(FrequencyRange=[0 1000],BackBaffled=true)` creates a back baffled omnidirectional microphone element with its frequency range specified between 0 and 1000 Hz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

FrequencyRange — Operating frequency range
[0 1e20] (default) | 1-by-2 real-valued row vector

Operating frequency range of the microphone element, specified as a 1-by-2 real-valued row vector in the form of [LowerBound HigherBound]. The microphone element has no response outside the specified frequency range. Units are in Hz.

Data Types: double

BackBaffled — Baffle the back direction of microphone element
false (default) | true

Baffle the back direction of microphone element, specified as false or true. When true, the microphone responses to all azimuth angles beyond ±90 degrees from broadside (zero degrees azimuth and elevation) are zero.

When the value of this property is false, the back direction of the microphone element is not baffled.

Data Types: logical

Usage

Syntax

```
RESP = microphone(FREQ,ANG)
```

Description

RESP = microphone(FREQ,ANG) returns the microphone’s magnitude response, RESP, at frequencies specified in FREQ and directions specified in ANG.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Input Arguments

FREQ — Signal frequencies
1-by-*P* row vector of positive values

Signal frequencies, specified as a 1-by-*P* row vector of positive values. Units are Hz.

ANG — Response directions
1-by-*Q* vector of real-values | 2-by-*Q* matrix of real-values

Response directions, specified as a 1-by-*Q* vector of real-values or a 2-by-*Q* matrix of real-values.

- If ANG is a 1-by-*Q* vector, each element specifies a direction’s azimuth angle. The azimuth angle must be between *–180°* and *180°*, inclusive. In this case, the corresponding elevation angle is assumed to be zero.

- If ANG is a 2-by-Q matrix, each column of the matrix specifies the direction in the form [azimuth;elevation]. The azimuth angle must be between *–180°* and *180°*, inclusive. The elevation angle must lie between *–90°* and *90°*, inclusive.

Units are in degrees.

Output Arguments

RESP — Microphone response
Q-by-*P* real-valued matrix

Microphone magnitude response, returned as an *Q*-by-*P* real-valued matrix. The matrix contains the responses of the microphone element at the *Q* angles specified in ANG and the *P* frequencies specified in FREQ.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named obj, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns
patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

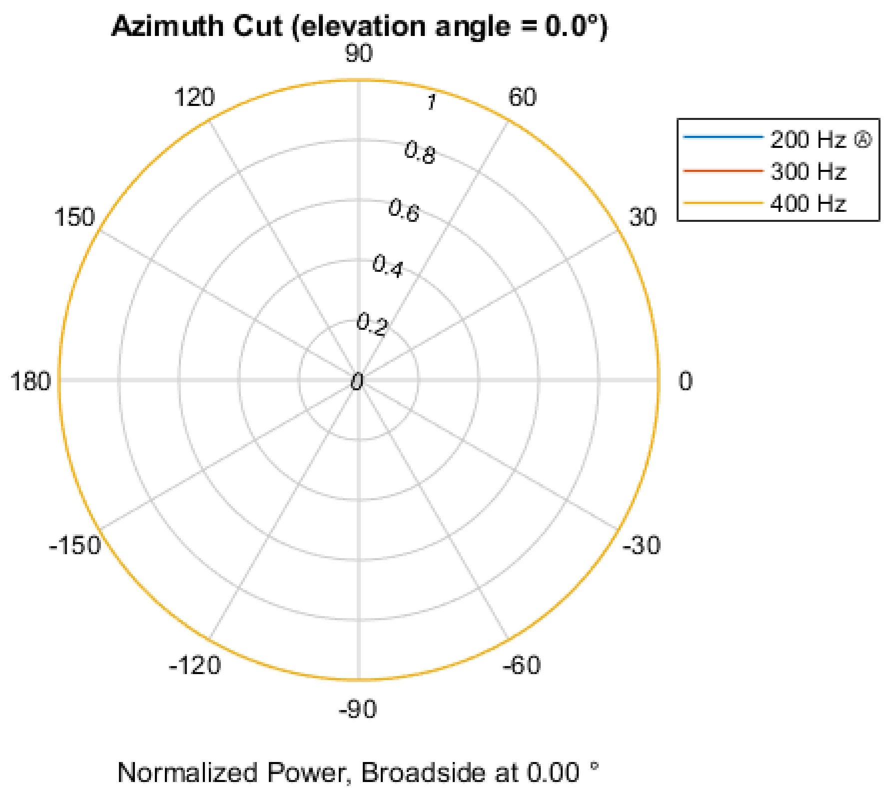
Display Omnidirectional Microphone Pattern

Create an omnidirectional microphone. Find the microphone response at 200, 300, and 400 Hz for the incident angle 0° azimuth and 0° elevation. Then, plot the azimuth response of the microphone at three frequencies.

```
microphone = phased.OmnidirectionalMicrophoneElement;
microphone.FrequencyRange=[20 2e3];
fc = [200 300 400];
ang = [0;0];
resp = microphone(fc,ang);
```

Plot the response pattern. The response patterns for at all three frequencies are the same.

```
pattern(microphone,fc,-180:180,0,'CoordinateSystem','polar','Type','power');
```



Directivity of Omnidirectional Microphone Element

Compute the directivity of an omnidirectional microphone element for several different directions.

Create the omnidirectional microphone element system object.

```
myMic = phased.OmnidirectionalMicrophoneElement();
```

Select the angles of interest at constant elevation angle set equal to zero degrees. Select seven azimuth angles centered at boresight (zero degrees azimuth and zero degrees elevation). Finally, set the desired frequency to 1 kHz.

```
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
freq = 1000;
```

Compute the directivity along the constant elevation cut.

```
d = directivity(myMic,freq,ang)
```

```
d = 7×1

     0
     0
```

```
     0
     0
     0
     0
     0
```

Next select the angles of interest to be at constant azimuth angle at zero degrees. All elevation angles are centered around boresight. The five elevation angles range from -20 to +20 degrees. Set the desired frequency to 1 GHz.

```
ang = [0,0,0,0,0; -20,-10,0,10,20];
freq = 1000;
```

Compute the directivity along the constant azimuth cut.

```
d = directivity(myMic,freq,ang)
```

```
d = 5×1

     0
     0
     0
     0
     0
```

For an omnidirectional microphone, the directivity is independent of direction.

Omnidirectional Microphone Element Does Not Support Polarization

Determine whether a `phased.OmnidirectionalMicrophoneElement` microphone element supports polarization.

```
microphone = phased.OmnidirectionalMicrophoneElement;
isPolarizationCapable(microphone)
```

```
ans = logical
   0
```

The returned value `0` shows that the omnidirectional microphone element does not support polarization.

Magnitude and Directivity Patterns of Omnidirectional Microphone

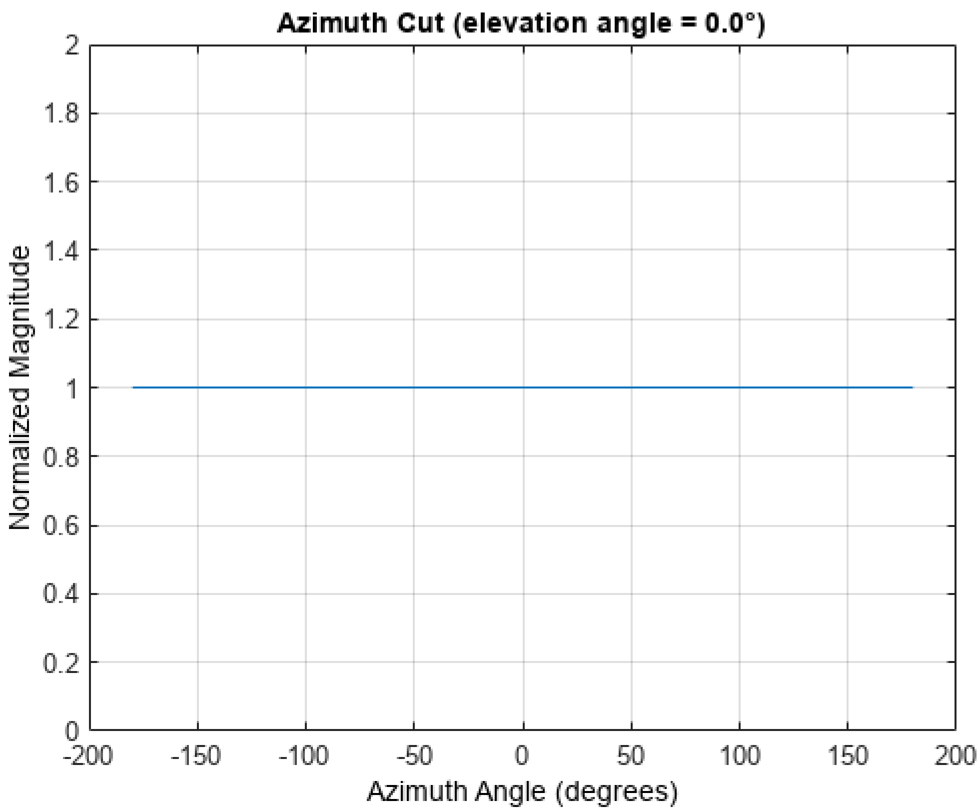
Construct an omnidirectional microphone and plot the magnitude and directivity patterns. The microphone operating frequency spans the range 20 to 20000 Hz.

Construct the omnidirectional microphone.

```
sOmni = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
```

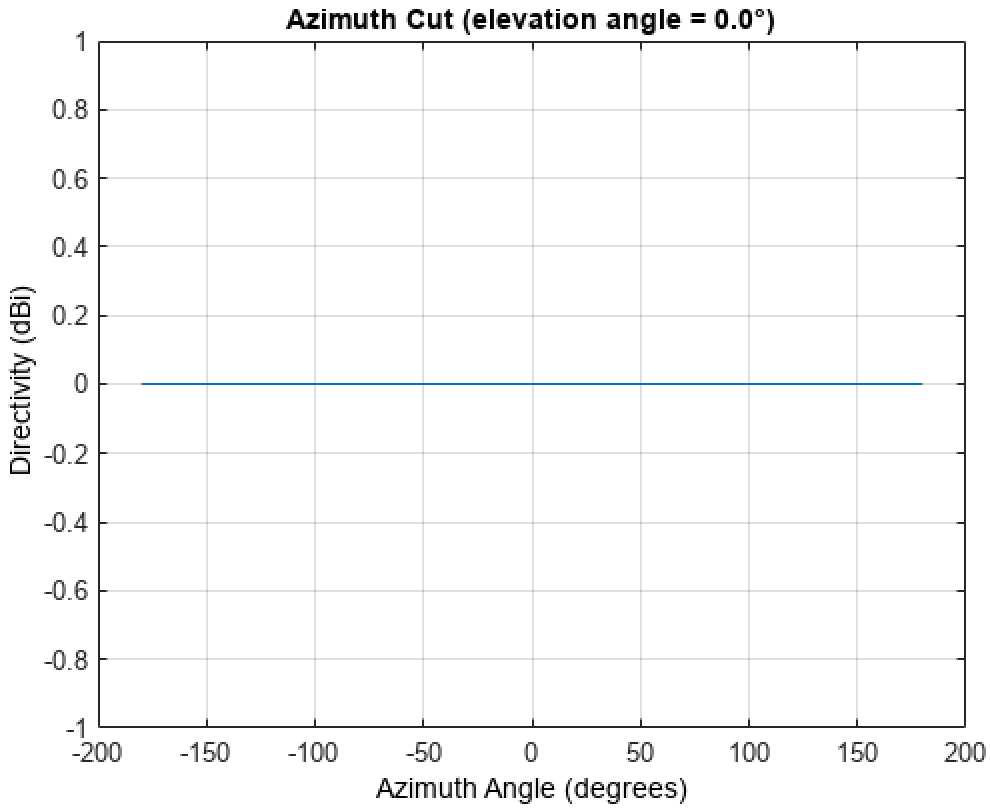
Plot the microphone magnitude pattern at 200 Hz.

```
fc = 200;
pattern(sOmni,fc,[-180:180],0,...
    'CoordinateSystem','rectangular',...
    'Type','efield')
```



Plot the microphone directivity.

```
pattern(sOmni,fc,[-180:180],0,...
    'CoordinateSystem','rectangular',...
    'Type','directivity')
```



The directivity is 0 dbi as expected for an omnidirectional element.

3-D Magnitude Pattern of Omnidirectional Microphone

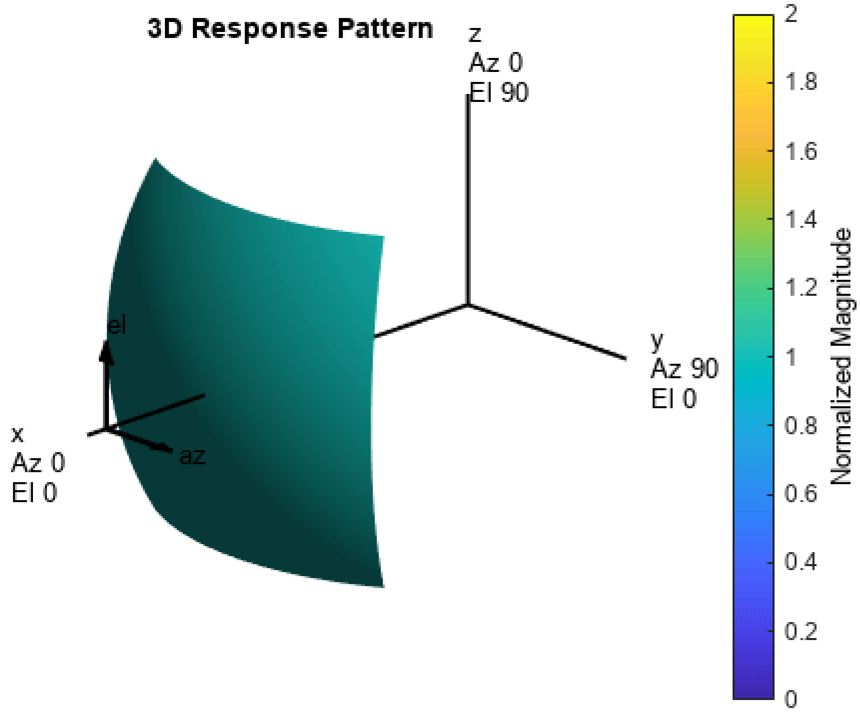
Construct an omnidirectional microphone with response in the frequency range 20-20000 Hz. Then, plot the 3-D magnitude pattern over a range of angles.

Construct the microphone element.

```
sOmin = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
```

Plot the 3-D pattern at 500 Hz between -30 to 30 degrees in both azimuth and elevation in 0.1 degree increments.

```
fc = 500;
pattern(sOmin,fc,[-30:0.1:30],[-30:0.1:30],...
    'CoordinateSystem','polar',...
    'Type','efield')
```



Azimuth Pattern of Omnidirectional Microphone Element

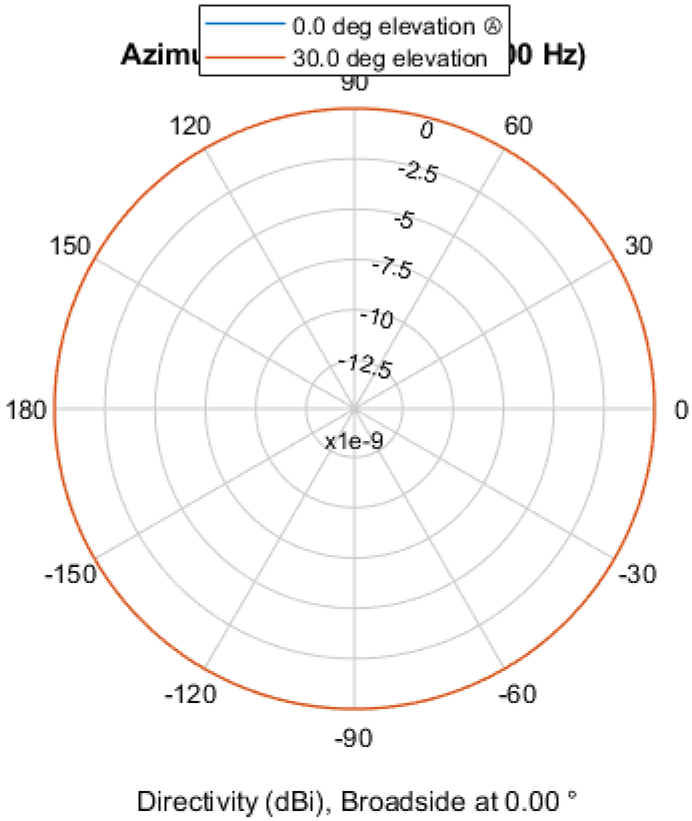
Create an omnidirectional microphone element. Plot an azimuth cut of the directivity at 0 and 30 degrees elevation. Assume an operating frequency of 500 Hz.

Create the microphone element.

```
sOmni = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[100,900]);
fc = 500;
```

Plot the azimuth pattern.

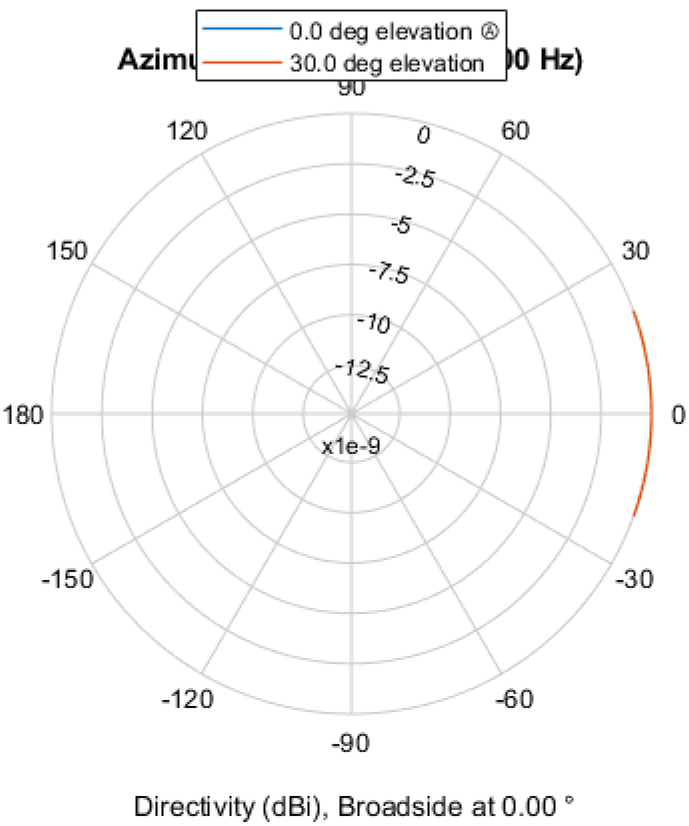
```
patternAzimuth(sOmni,fc,[0 30])
```



Because of the omnidirectionality of the microphone, the two patterns coincide.

Plot a reduced range of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(sOmni,fc,[0 30],'Azimuth',[-20:20])
```



Elevation Pattern of Omnidirectional Microphone Element

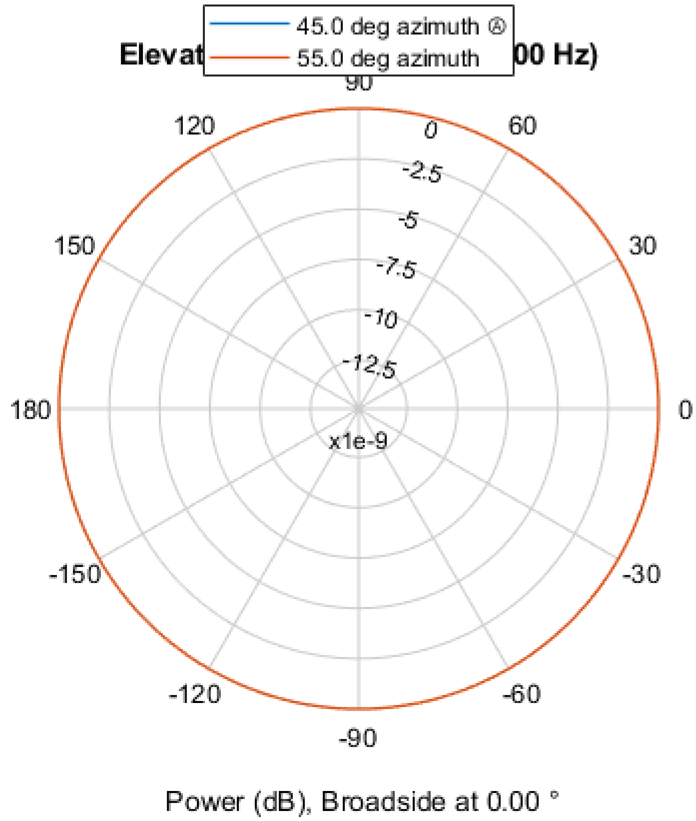
Construct an omnidirectional microphone element. Plot an elevation cut of the power 45 and 55 degrees azimuth. Assume the operating frequency is 500 Hz.

Create the microphone element.

```
fc = 500;
sOmni = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[100,900]);
```

Display the power pattern.

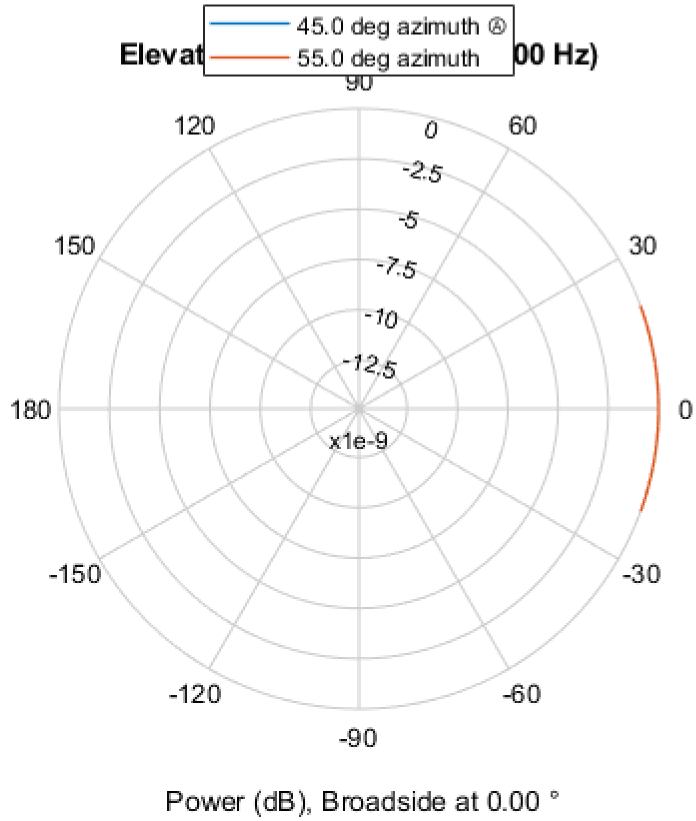
```
patternElevation(sOmni,fc,[45 55],'Type','powerdb')
```



Because of the omnidirectionality, the two plots coincide.

Plot a reduced range of elevation angles using the `Elevation` parameter.

```
patternElevation(sOmni,fc,[45 55],...
    'Elevation',[-20:20],...
    'Type','powerdb')
```



Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `pattern`, `patternAzimuth`, and `patternElevation` methods are not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.IsotropicProjector` | `phased.IsotropicHydrophone` | `phased.CustomMicrophoneElement` | `phased.ULA` | `phased.URA` | `phased.ConformalArray` | `uv2azel` | `phitheta2azel`

directivity

System object: `phased.OmnidirectionalMicrophoneElement`
Package: `phased`

Directivity of omnidirectional microphone element

Syntax

`D = directivity(H,FREQ,ANGLE)`

Description

`D = directivity(H,FREQ,ANGLE)` returns the "Directivity (dBi)" on page 1-1108 of an omnidirectional microphone element, `H`, at frequencies specified by `FREQ` and in direction angles specified by `ANGLE`.

Input Arguments

`H` — Omnidirectional Microphone Element

System object

Omnidirectional microphone element specified as a `phased.OmnidirectionalMicrophoneElement` System object.

Example: `H = phased.OmnidirectionalMicrophoneElement`

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`ANGLE` — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If ANGLE is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: [45 60; 0 10]

Data Types: double

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by ANGLE. Each column corresponds to one of the *L* frequency values specified in FREQ. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Omnidirectional Microphone Element

Compute the directivity of an omnidirectional microphone element for several different directions.

Create the omnidirectional microphone element system object.

```
myMic = phased.OmnidirectionalMicrophoneElement();
```

Select the angles of interest at constant elevation angle set equal to zero degrees. Select seven azimuth angles centered at boresight (zero degrees azimuth and zero degrees elevation). Finally, set the desired frequency to 1 kHz.

```
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
freq = 1000;
```

Compute the directivity along the constant elevation cut.

```
d = directivity(myMic,freq,ang)
```

```
d = 7×1

     0
     0
     0
     0
     0
     0
     0
```

Next select the angles of interest to be at constant azimuth angle at zero degrees. All elevation angles are centered around boresight. The five elevation angles range from -20 to +20 degrees. Set the desired frequency to 1 GHz.

```
ang = [0,0,0,0,0; -20,-10,0,10,20];
freq = 1000;
```

Compute the directivity along the constant azimuth cut.

```
d = directivity(myMic,freq,ang)
```

```
d = 5×1

     0
     0
     0
     0
     0
```

For an omnidirectional microphone, the directivity is independent of direction.

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`patternElevation` | `patternAzimuth` | `pattern`

isPolarizationCapable

System object: `phased.OmnidirectionalMicrophoneElement`
Package: `phased`

Polarization capability

Syntax

```
flag = isPolarizationCapable(microphone)
```

Description

`flag = isPolarizationCapable(microphone)` returns a Boolean value, `flag`, indicating whether the `phased.OmnidirectionalMicrophoneElement` supports polarization. An element supports polarization if it can create or respond to polarized fields. This microphone element, as all microphone elements, does not support polarization.

Input Arguments

`microphone` — Omni-directional microphone element

Omni-directional microphone element specified as a `phased.OmnidirectionalMicrophoneElement` System object

Output Arguments

`flag` — Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the microphone element supports polarization or `false` if it does not. Because the `phased.OmnidirectionalMicrophoneElement` object does not support polarization, `flag` is always returned as `false`.

Examples

Omnidirectional Microphone Element Does Not Support Polarization

Determine whether a `phased.OmnidirectionalMicrophoneElement` microphone element supports polarization.

```
microphone = phased.OmnidirectionalMicrophoneElement;
isPolarizationCapable(microphone)
```

```
ans = logical
   0
```

The returned value `0` shows that the omnidirectional microphone element does not support polarization.

pattern

System object: `phased.OmnidirectionalMicrophoneElement`
Package: `phased`

Plot omnidirectional microphone element directivity and patterns

Syntax

```
pattern(sElem,FREQ)
pattern(sElem,FREQ,AZ)
pattern(sElem,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sElem,FREQ)` plots the 3-D array directivity pattern (in dBi) for the element specified in `sElem`. The operating frequency is specified in `FREQ`.

`pattern(sElem,FREQ,AZ)` plots the element directivity pattern at the specified azimuth angle.

`pattern(sElem,FREQ,AZ,EL)` plots the element directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the element pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See “Convert plotResponse to pattern” on page 1-1118 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sElem` — Omnidirectional microphone element

System object

Omnidirectional microphone element, specified as a `phased.OmnidirectionalMicrophoneElement` System object.

Example: `sElem = phased.OmnidirectionalMicrophoneElement;`

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the

pattern azimuth and elevation, respectively. AZ values must lie between –180° and 180°. EL values must lie between –90° and 90°. If 'CoordinateSystem' is set to 'uv', AZ and EL then specify *U* and *V* coordinates, respectively. AZ and EL must lie between -1 and 1.

Example: 'uv'

Data Types: char

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

PlotStyle — Plotting style
'overlay' (default) | 'waterfall'

Plotting style, specified as the comma-separated pair consisting of 'Plotstyle' and either 'overlay' or 'waterfall'. This parameter applies when you specify multiple frequencies in FREQ in 2-D plots. You can draw 2-D plots by setting one of the arguments AZ or EL to a scalar.

Data Types: char

Output Arguments

PAT — Element pattern
N-by-*M* real-valued matrix

Element pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of PAT correspond to the azimuth angles in the vector specified by EL_ANG. The columns correspond to the elevation angles in the vector specified by AZ_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

Examples

Magnitude and Directivity Patterns of Omnidirectional Microphone

Construct an omnidirectional microphone and plot the magnitude and directivity patterns. The microphone operating frequency spans the range 20 to 20000 Hz.

Construct the omnidirectional microphone.

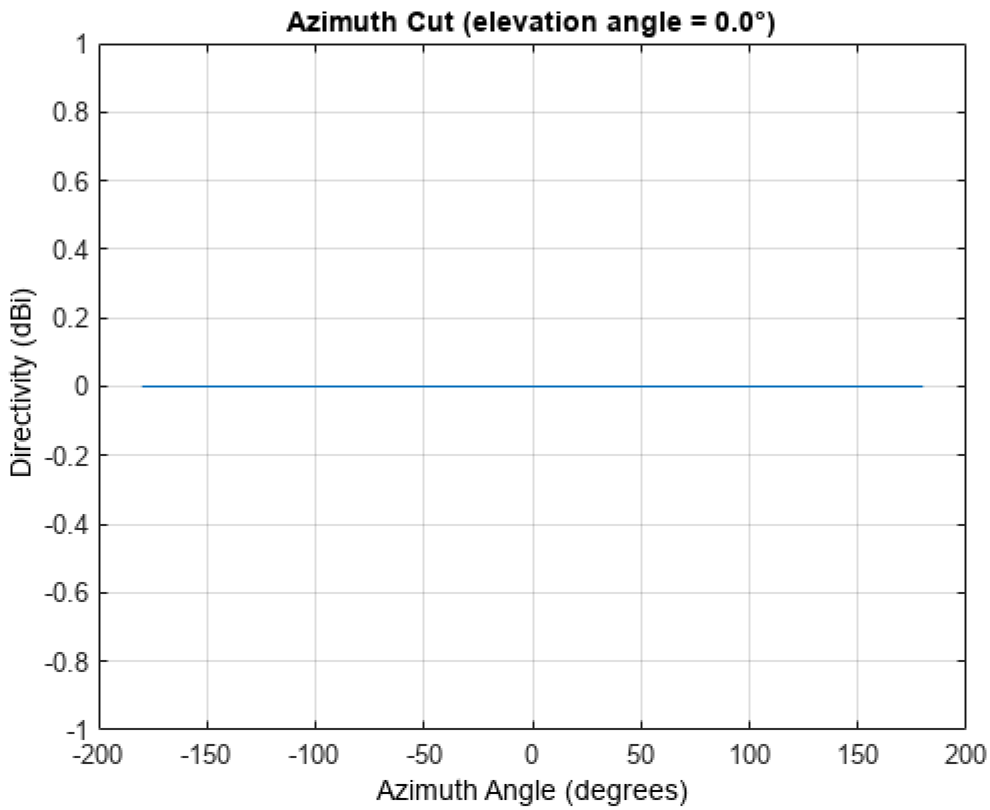
```
sOmni = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
```

Plot the microphone magnitude pattern at 200 Hz.

```
fc = 200;
pattern(sOmni,fc,[-180:180],0,...
    'CoordinateSystem','rectangular',...
    'Type','efield')
```

Plot the microphone directivity.

```
pattern(sOmni,fc,[-180:180],0,...
    'CoordinateSystem','rectangular',...
    'Type','directivity')
```



The directivity is 0 dbi as expected for an omnidirectional element.

3-D Magnitude Pattern of Omnidirectional Microphone

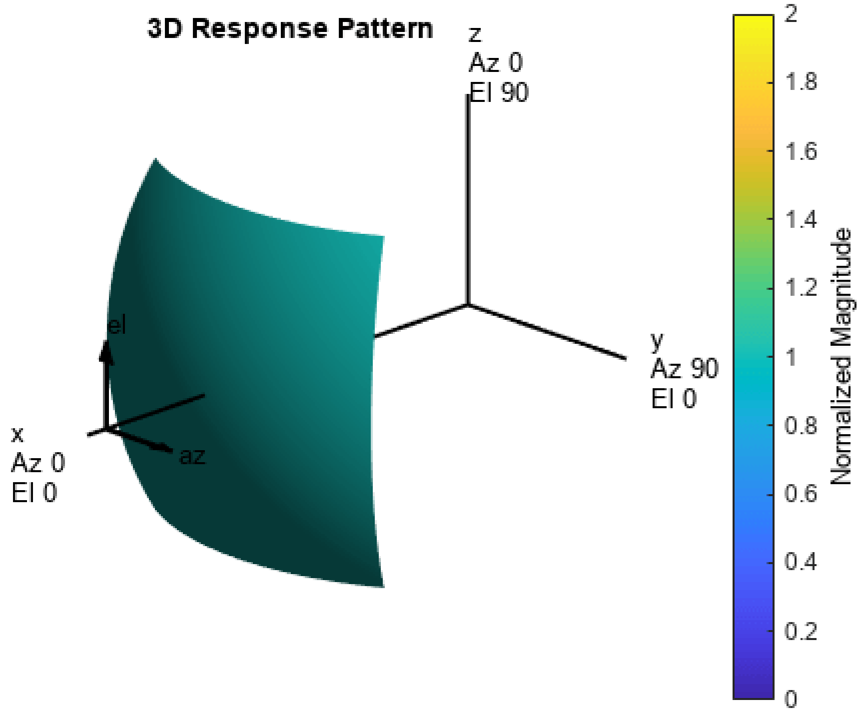
Construct an omnidirectional microphone with response in the frequency range 20-20000 Hz. Then, plot the 3-D magnitude pattern over a range of angles.

Construct the microphone element.

```
sOmin = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
```

Plot the 3-D pattern at 500 Hz between -30 to 30 degrees in both azimuth and elevation in 0.1 degree increments.

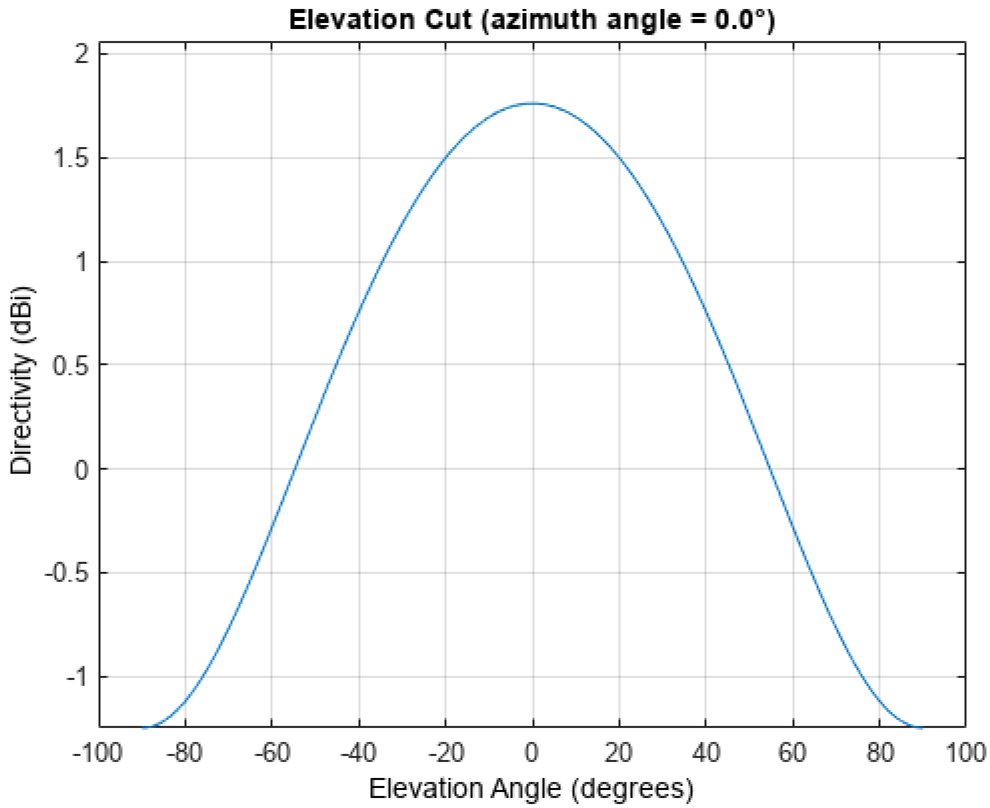
```
fc = 500;
pattern(sOmin,fc,[-30:0.1:30],[-30:0.1:30],...
    'CoordinateSystem','polar',...
    'Type','efield')
```



Plot Directivity of Crossed-Dipole Antenna

Create a crossed-dipole antenna. Assume the antenna works between 1 and 2 GHz and its operating frequency is 1.5 GHz. Then, plot the directivity at a constant azimuth of 0˚.

```
antenna = phased.CrossedDipoleAntennaElement('FrequencyRange',[1e9 2e9]);
fc = 1.5e9;
pattern(antenna,fc,0,-90:90,'Type','directivity', ...
    'CoordinateSystem','rectangular')
```



The directivity is maximum at 0° elevation and attains a value of approximately 1.75 dB.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td><code>'Format'</code> and <code>'RespCut'</code> name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using <code>plotResponse</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'RespCut'</code> to <code>'Az'</code> or <code>'El'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using either the <code>'AzimuthAngles'</code> or <code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'RespCut'</code> to <code>'3D'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using both the <code>'AzimuthAngles'</code> and<code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'RespCut'</code> to<code>'U'</code>. Set <code>'Format'</code> to <code>'UV'</code>. Set the display range using the <code>'UGrid'</code> name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'RespCut'</code> to<code>'3D'</code>. Set <code>'Format'</code> to</td></tr>
</table>
</td>
<td><code>'CoordinateSystem'</code> name-value pair used together with the <code>AZ</code> and <code>EL</code> input arguments.

<code>'CoordinateSystem'</code> has the same options as the <code>plotResponse</code> method <code>'Format'</code>name-value pair, except that <code>'line'</code> is now named <code>'rectangular'</code>. The table shows how to create different types of plots using <code>pattern</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify either <code>AZ</code> or <code>EL</code> as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify both <code>AZ</code> and <code>EL</code> as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space vector.</td></tr>
</table>
If you set <code>CoordinateSystem</code> to <code>'uv'</code>, enter the UV grid values using <code>AZ</code> and <code>EL</code>.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternElevation` | `patternAzimuth`

patternAzimuth

System object: `phased.OmnidirectionalMicrophoneElement`
Package: `phased`

Plot omnidirectional microphone element directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sElem,FREQ)
patternAzimuth(sElem,FREQ,EL)
patternAzimuth(sElem,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sElem,FREQ)` plots the 2-D element directivity pattern versus azimuth (in dBi) for the element `sElem` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

`patternAzimuth(sElem,FREQ,EL)`, in addition, plots the 2-D element directivity pattern versus azimuth (in dBi) at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sElem,FREQ,EL,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sElem` — Omnidirectional microphone element
System object

Omnidirectional microphone element, specified as a `phased.OmnidirectionalMicrophoneElement` System object.

Example: `sElem = phased.OmnidirectionalMicrophoneElement;`

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

Examples

Azimuth Pattern of Omnidirectional Microphone Element

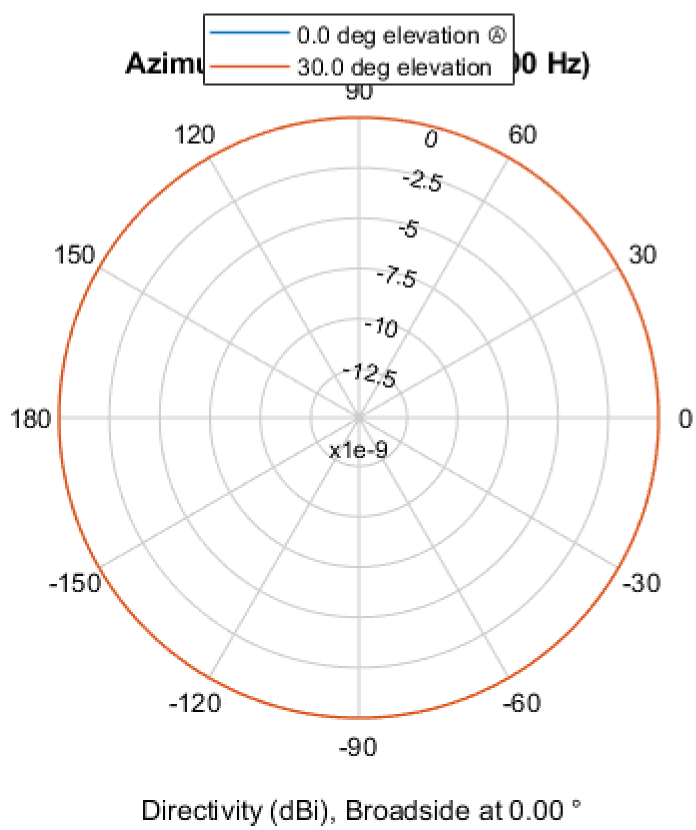
Create an omnidirectional microphone element. Plot an azimuth cut of the directivity at 0 and 30 degrees elevation. Assume an operating frequency of 500 Hz.

Create the microphone element.

```
sOmni = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[100,900]);
fc = 500;
```

Plot the azimuth pattern.

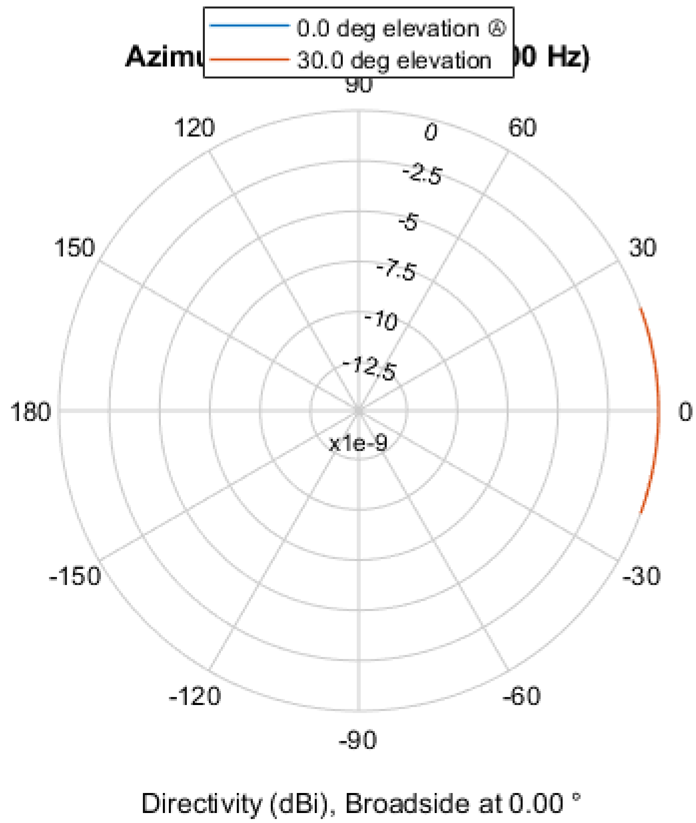
```
patternAzimuth(sOmni,fc,[0 30])
```



Because of the omnidirectionality of the microphone, the two patterns coincide.

Plot a reduced range of azimuth angles using the `Azimuth` parameter.

```
patternAzimuth(sOmni,fc,[0 30],'Azimuth',[-20:20])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or

array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`patternElevation` | `pattern`

patternElevation

System object: phased.OmnidirectionalMicrophoneElement
Package: phased

Plot omnidirectional microphone element directivity or pattern versus elevation

Syntax

```
patternElevation(sElem,FREQ)
patternElevation(sElem,FREQ,AZ)
patternElevation(sElem,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

patternElevation(sElem,FREQ) plots the 2-D element directivity pattern versus elevation (in dBi) for the element sElem at zero degrees azimuth angle. The argument FREQ specifies the operating frequency.

patternElevation(sElem,FREQ,AZ), in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by AZ. When AZ is a vector, multiple overlaid plots are created.

patternElevation(sElem,FREQ,AZ,Name,Value) plots the element pattern with additional options specified by one or more Name,Value pair arguments.

PAT = patternElevation(___) returns the element pattern. PAT is a matrix whose entries represent the pattern at corresponding sampling points specified by the 'Elevation' parameter and the AZ input argument.

Input Arguments

sElem — Omnidirectional microphone element

System object

Omnidirectional microphone element, specified as a phased.OmnidirectionalMicrophoneElement System object.

Example: sElem = phased.OmnidirectionalMicrophoneElement;

FREQ — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

Elevation — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Elevation Pattern of Omnidirectional Microphone Element

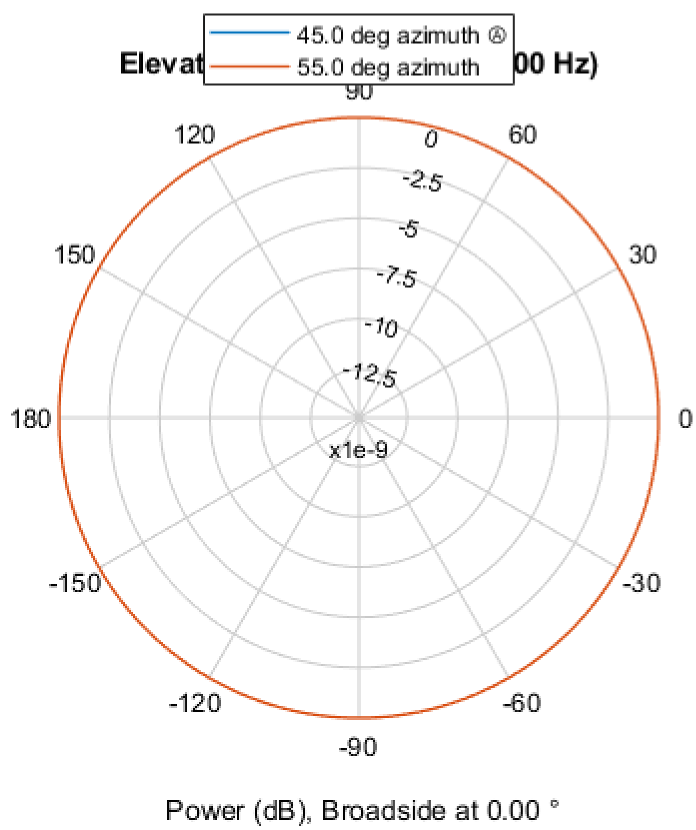
Construct an omnidirectional microphone element. Plot an elevation cut of the power 45 and 55 degrees azimuth. Assume the operating frequency is 500 Hz.

Create the microphone element.

```
fc = 500;
sOmni = phased.OmnidirectionalMicrophoneElement('FrequencyRange',[100,900]);
```

Display the power pattern.

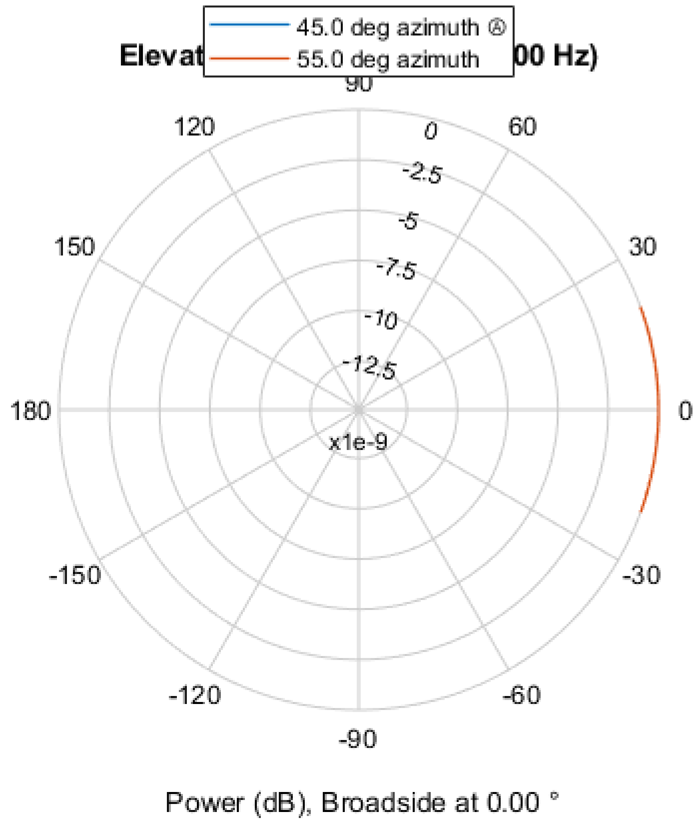
```
patternElevation(sOmni,fc,[45 55],'Type','powerdb')
```



Because of the omnidirectionality, the two plots coincide.

Plot a reduced range of elevation angles using the `Elevation` parameter.

```
patternElevation(sOmni,fc,[45 55],...
    'Elevation',[-20:20],...
    'Type','powerdb')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `pattern`

plotResponse

System object: `phased.OmnidirectionalMicrophoneElement`
Package: `phased`

Plot response pattern of microphone

Syntax

```
plotResponse(H,FREQ)
plotResponse(H,FREQ,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ)` plots the element response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`.

`plotResponse(H,FREQ,Name,Value)` plots the element response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Element System object

FREQ

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

CutAngle

Cut angle specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

Format

Format of the plot, using one of 'Line', 'Polar', or 'UV'. If you set Format to 'UV', FREQ must be a scalar.

Default: 'Line'

NormalizeResponse

Set this value to true to normalize the response pattern. Set this value to false to plot the response pattern without normalizing it. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: true

OverlayFreq

Set this value to true to overlay pattern cuts in a 2-D line plot. Set this value to false to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is false, FREQ must be a vector with at least two entries.

This parameter applies only when Format is not 'Polar' and RespCut is not '3D'.

Default: true

Polarization

Specify the polarization options for plotting the antenna response pattern. The allowable values are |'None' | 'Combined' | 'H' | 'V' | where

- 'None' specifies plotting a nonpolarized response pattern
- 'Combined' specifies plotting a combined polarization response pattern
- 'H' specifies plotting the horizontal polarization response pattern
- 'V' specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is 'None'. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: 'None'

RespCut

Cut of the response. Valid values depend on Format, as follows:

- If Format is 'Line' or 'Polar', the valid values of RespCut are 'Az', 'El', and '3D'. The default is 'Az'.
- If Format is 'UV', the valid values of RespCut are 'U' and '3D'. The default is 'U'.

If you set RespCut to '3D', FREQ must be a scalar.

Unit

The unit of the plot. Valid values are 'db', 'mag', 'pow', or 'dbi'. This parameter determines the type of plot that is produced.

Unit value	Plot type
db	power pattern in dB scale
mag	field pattern
pow	power pattern
dbi	directivity

Default: 'db'

AzimuthAngles

Azimuth angles for plotting element response, specified as a row vector. The AzimuthAngles parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'Az' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the AzimuthAngles and ElevationAngles parameters simultaneously.

Default: [-180:180]

ElevationAngles

Elevation angles for plotting element response, specified as a row vector. The ElevationAngles parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'El' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the ElevationAngles and AzimuthAngles parameters simultaneously.

Default: [-90:90]

UGrid

U coordinate values for plotting element response, specified as a row vector. The UGrid parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to 'U' or '3D'. The values of UGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the UGrid and VGrid parameters simultaneously.

Default: [-1:0.01:1]

VGrid

V coordinate values for plotting element response, specified as a row vector. The VGrid parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to '3D'. The values of VGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the VGrid and UGrid parameters simultaneously.

Default: [-1:0.01:1]

Examples

Plot Response and Directivity of Omnidirectional Microphone

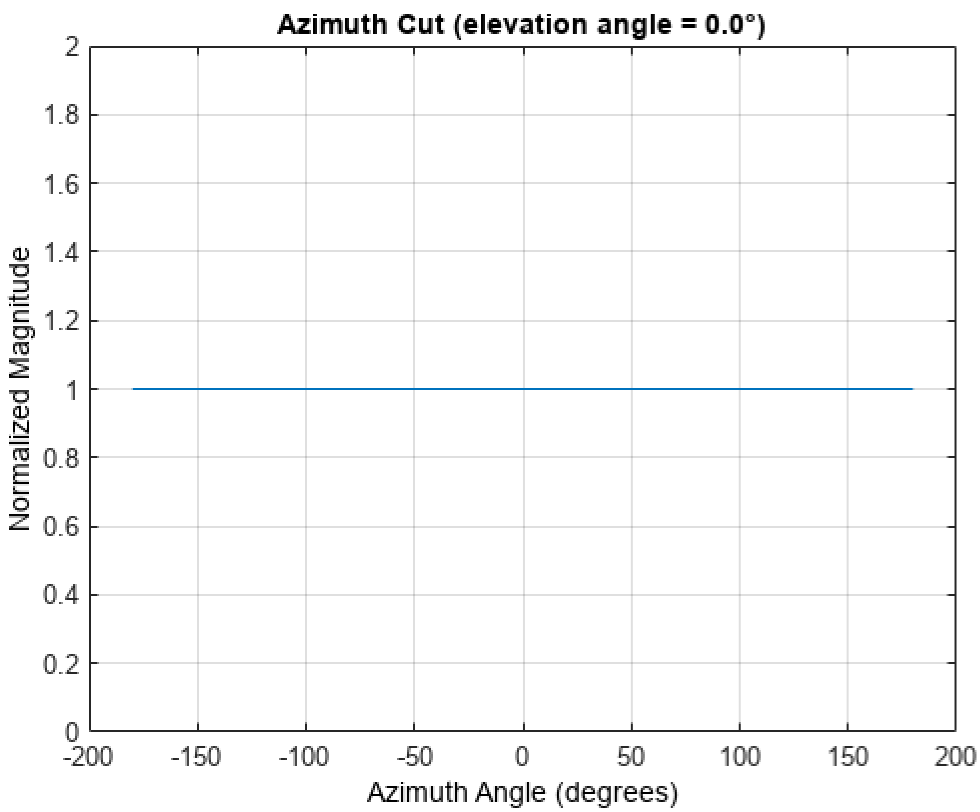
This example shows how to construct an omnidirectional microphone and how to plot its response and directivity. The microphone operating frequency spans the range 20 to 20000 Hz.

Construct the omnidirectional microphone.

```
sOmni = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
```

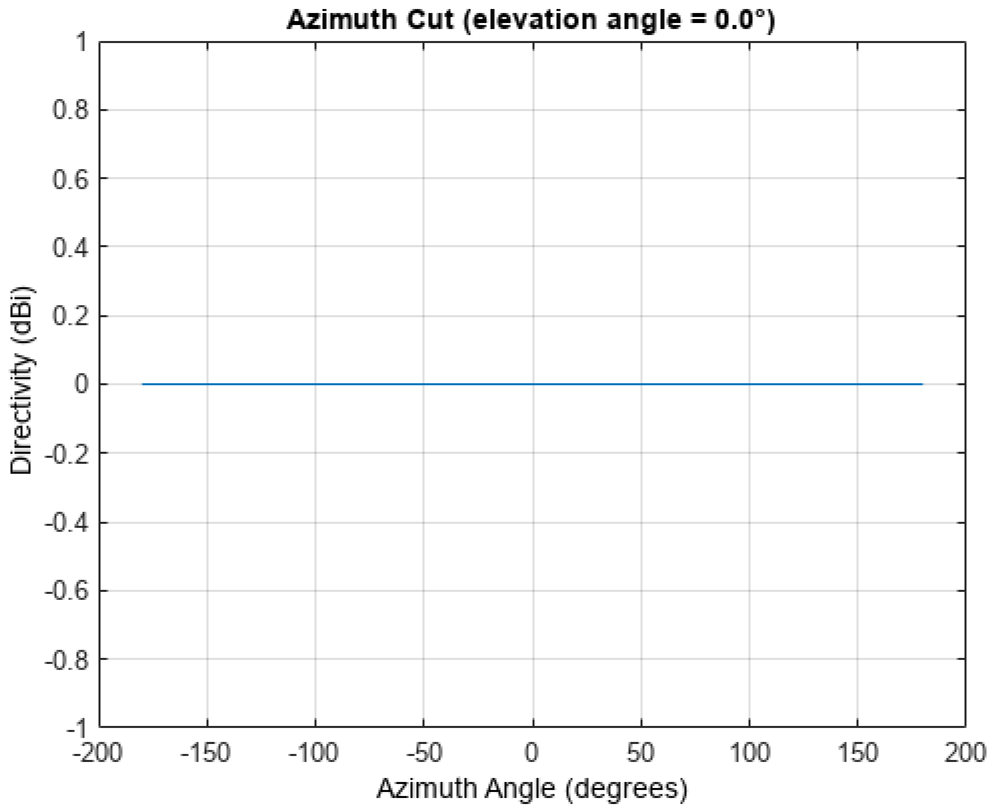
Plot the microphone response at 200 Hz.

```
fc = 200;
plotResponse(sOmni,fc,'Unit','mag');
```



Plot the microphone directivity.

```
plotResponse(sOmni,fc,'Unit','dbi');
```



Plot 3-D Response of Omnidirectional Microphone

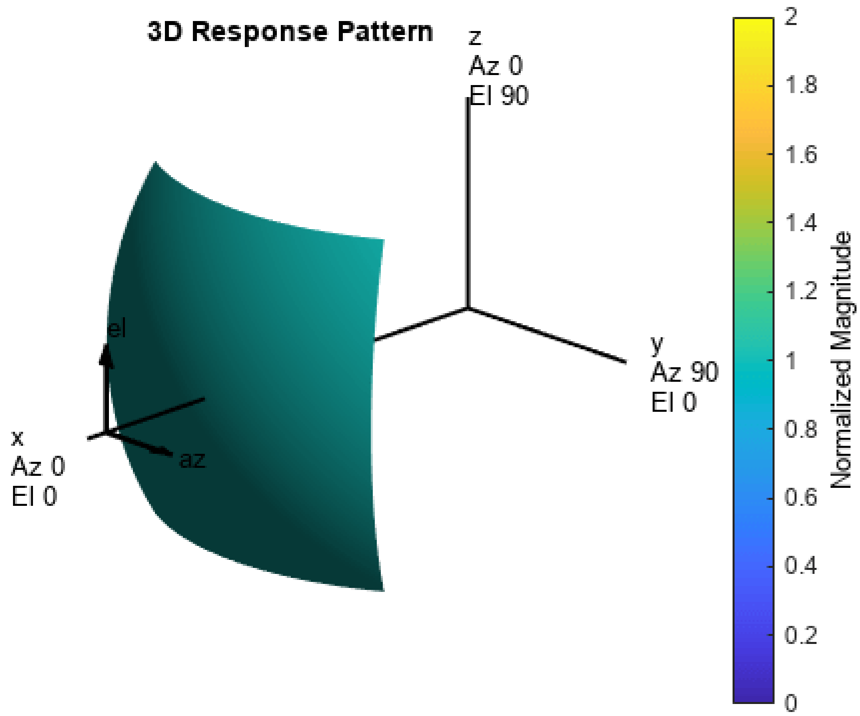
This example shows how to construct an omnidirection microphone with response in the frequency range 20 - 20000 Hz and how to plot its 3-D response over a range of angles.

Construct the microphone element.

```
sOmin = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
```

Plot the 3-D response at 500 Hz. Show the response between -30 to 30 degrees in both azimuth and elevation in 0.1 degree increments.

```
plotResponse(sOmin,500,'Format','Polar',...
    'RespCut','3D','Unit','mag',...
    'AzimuthAngles',[-30:0.1:30],...
    'ElevationAngles',[-30:0.1:30]);
```



See Also

`uv2azel` | `azel2uv`

step

System object: `phased.OmnidirectionalMicrophoneElement`
Package: `phased`

Output response of microphone

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the microphone's magnitude response, `RESP`, at frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`H`

Microphone object.

`FREQ`

Frequencies in hertz. `FREQ` is a row vector of length L.

`ANG`

Directions in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If `ANG` is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

Output Arguments

RESP

Response of microphone. RESP is an M-by-L matrix that contains the responses of the microphone element at the M angles specified in ANG and the L frequencies specified in FREQ.

Examples

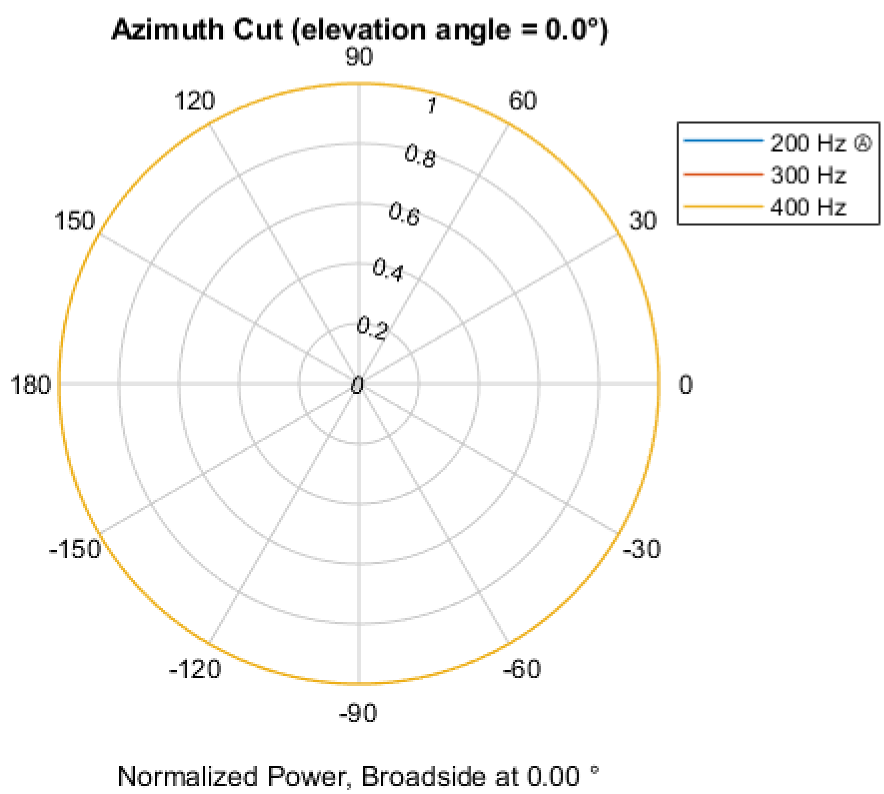
Display Omnidirectional Microphone Pattern

Create an omnidirectional microphone. Find the microphone response at 200, 300, and 400 Hz for the incident angle 0° azimuth and 0° elevation. Then, plot the azimuth response of the microphone at three frequencies.

```
microphone = phased.OmnidirectionalMicrophoneElement;
microphone.FrequencyRange=[20 2e3];
fc = [200 300 400];
ang = [0;0];
resp = microphone(fc,ang);
```

Plot the response pattern. The response patterns for at all three frequencies are the same.

```
pattern(microphone,fc,-180:180,0,'CoordinateSystem','polar','Type','power');
```



See Also

uv2azel | phitheta2azel

phased.PartitionedArray

Package: phased

Phased array partitioned into subarrays

Description

The PartitionedArray object represents a phased array that is partitioned into one or more subarrays.

To obtain the response of the subarrays in a partitioned array:

1. Define and set up your partitioned array. See “Construction” on page 1-1141.
2. Call step to compute the response of the subarrays according to the properties of phased.PartitionedArray. The behavior of step is specific to each object in the toolbox.

You can also specify a PartitionedArray object as the value of the SensorArray or Sensor property of objects that perform beamforming, steering, and other operations.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.PartitionedArray creates a partitioned array System object, H. This object represents an array that is partitioned into subarrays.

H = phased.PartitionedArray(Name,Value) creates a partitioned array object, H, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

Array

Sensor array

Sensor array, specified as any array System object belonging to Phased Array System Toolbox.

Default: phased.ULA('NumElements',4)

SubarraySelection

Subarray definition matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the number of elements in the array. Each row of the matrix corresponds to a subarray and each entry in the row indicates whether or not an element belongs to the subarray. When the entry is zero, the element does not belong to the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray is at the subarray geometric center. The `SubarraySelection` and `Array` properties determine the geometric center.

Default: `[1 1 0 0; 0 0 1 1]`

`SubarraySteering`

Subarray steering method

Specify the method of subarray steering as either `'None'` | `'Phase'` | `'Time'` | `'Custom'`.

- When you set this property to `'Phase'`, a phase shifter is used to steer the subarray. Use the `STEERANG` argument of the `step` method to define the steering direction.
- When you set this property to `'Time'`, subarrays are steered using time delays. Use the `STEERANG` argument of the `step` method to define the steering direction.
- When you set this property to `'Custom'`, subarrays are steered by setting independent weights for all elements in each subarray. Use the `WS` argument of the `step` method to define the weights for all subarrays.

Default: `'None'`

`PhaseShifterFrequency`

Subarray phase shifter frequency

Specify the operating frequency of phase shifters that perform subarray steering. The property value is a positive scalar in hertz. This property applies when you set the `SubarraySteering` property to `'Phase'`.

Default: `300e6`

`NumPhaseShifterBits`

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Default: `0`

Methods

Specific to `phased.PartitionedArray` Object	
`beamwidth`	Compute and display beamwidth for a subarray
`collectPlaneWave`	Simulate received plane waves
`directivity`	Directivity of partitioned array

Specific to phased.PartitionedArray Object	
getElementPosition	Positions of array elements
getNumElements	Number of elements in array
getNumSubarrays	Number of subarrays in array
getSubarrayPosition	Positions of subarrays in array
isPolarizationCapable	Polarization capability
pattern	Plot partitioned array directivity, field, and power patterns
patternAzimuth	Plot partitioned array directivity or pattern versus azimuth
patternElevation	Plot partitioned array directivity or pattern versus elevation
plotResponse	Plot response pattern of array
step	Output responses of subarrays
viewArray	View array geometry

Common to All System Objects	
release	Allow System object property value changes

Examples

Azimuth Response of Partitioned ULA

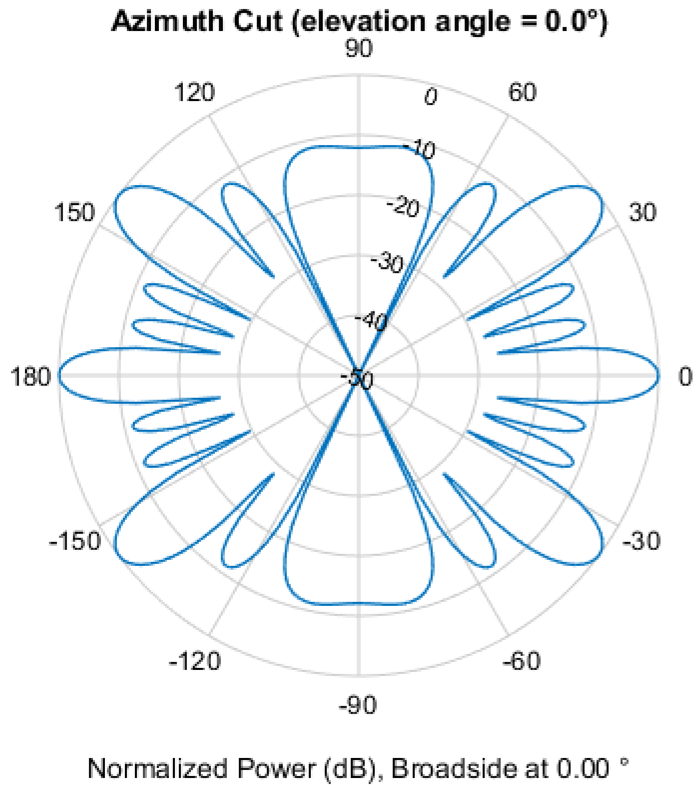
Plot the azimuth response of a 4-element ULA partitioned into two 2-element ULA's. The element spacing is one-half wavelength.

Create the ULA, and partition it into two 2-element ULA's.

```
sULA = phased.ULA('NumElements',4,'ElementSpacing',0.5);
sPA = phased.PartitionedArray('Array',sULA,...
    'SubarraySelection',[1 1 0 0;0 0 1 1]);
```

Plot the azimuth response of the array. Assume the operating frequency is 1 GHz and the propagation speed is the speed of light.

```
fc = 1e9;
pattern(sPA,fc,[-180:180],0,'Type','powerdb',...
    'CoordinateSystem','polar',...
    'Normalize',true)
```



Response of Subarrays of Partitioned ULA

Create a 4-element ULA. Then partition the ULA into two 2-element ULAs. Then, calculate the response at boresight of a 4-element ULA partitioned into two 2-element ULAs.

```
sULA = phased.ULA('NumElements',4,'ElementSpacing',0.5);
sPA = phased.PartitionedArray('Array',sULA,...
   'SubarraySelection',[1 1 0 0;0 0 1 1]);
```

Calculate the response at 1 GHz. The signal propagation speed is the speed of light.

```
fc = 1e9;
resp = step(sPA,fc,[0;0],physconst('LightSpeed'))
```

```
resp = 2×1

     2
     2
```

Subarray Element Weights for Partitioned Array

Create a partitioned URA array with three subarrays of different sizes. The subarrays have 8, 16, and 32 elements. Use different sets of subarray element weights for each subarray.

Create a 4-by-56 element URA.

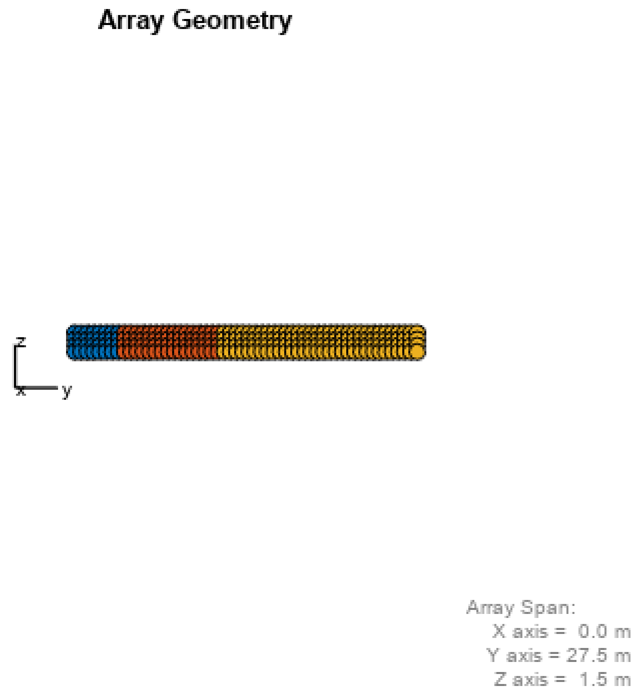
```
antenna = phased.IsotropicAntennaElement;
fc = 300e6;
c = physconst('LightSpeed');
lambda = c/fc;
n1 = 2^3;
n2 = 2^4;
n3 = 2^5;
nrows = 4;
ncols = n1 + n2 + n3;
array = phased.URA('Element',antenna,'Size',[nrows,ncols]);
```

Select the three subarrays by setting the selection matrix.

```
sel1 = zeros(nrows,ncols);
sel2 = sel1;
sel3 = sel1;
sel = zeros(3,nrows*ncols);
for r = 1:nrows
    sel1(r,1:n1) = 1;
    sel2(r,(n1+1):(n1+n2)) = 1;
    sel3(r,((n1+n2)+1):ncols) = 1;
end
sel(1,:) = sel1(:);
sel(2,:) = sel2(:);
sel(3,:) = sel3(:);
```

Create the partitioned array.

```
partarray = phased.PartitionedArray('Array',array, ...
    'SubarraySelection',sel,'SubarraySteering','Custom');
viewArray(partarray,'ShowSubarray','All');
```



Set weights for each subarray and get the response of each subarray. Put the weights in a cell array.

```
wts1 = ones(nrows*n1,1);
wts2 = 1.5*ones(nrows*n2,1);
wts3 = 3*ones(nrows*n3,1);
resp = partarray(fc,[30;0],c,{wts1,wts2,wts3})
```

```
resp = 3×1 complex

   0.0246 + 0.0000i
   0.0738 - 0.0000i
   0.2951 - 0.0000i
```

Version History

Introduced in R2012a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, `plotResponse`, and `viewArray` methods are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects
`phased.ULA` | `phased.URA` | `phased.UCA` | `phased.ConformalArray` | `phased.ReplicatedSubarray`

Apps
Sensor Array Analyzer

Topics
Subarrays in Phased Array Antennas
Phased Array Gallery
“Subarrays Within Arrays”

directivity

System object: phased.PartitionedArray
Package: phased

Directivity of partitioned array

Syntax

```
D = directivity(H,FREQ,ANGLE)
D = directivity(H,FREQ,ANGLE,Name,Value)
```

Description

D = directivity(H,FREQ,ANGLE) returns the "Directivity" on page 1-1151 of a partitioned array of antenna or microphone elements, H, at frequencies specified by FREQ and in angles of direction specified by ANGLE.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

D = directivity(H,FREQ,ANGLE,Name,Value) returns the directivity with additional options specified by one or more Name,Value pair arguments.

Input Arguments

H — Partitioned array

System object

Partitioned array, specified as a phased.PartitionedArray System object.

Example: H = phased.PartitionedArray;

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.
- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: [1e8 2e6]

Data Types: double

`ANGLE` — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PropagationSpeed` — Signal propagation speed

speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Subarray weights

1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Subarray weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*M* complex-valued matrix. The dimension *N* is the number of subarrays in the array. The dimension *L* is the number of frequencies specified by the `FREQ` argument.

`Weights` dimension	`FREQ` dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in the `FREQ` argument.

Example: `'Weights',ones(N,M)`

Data Types: `double`

`SteerAngle` — Subarray steering angle
`[0;0]` (default) | scalar | 2-element column vector

Subarray steering angle, specified as the comma-separated pair consisting of `'SteerAngle'` and a scalar or a 2-by-1 column vector.

If `'SteerAngle'` is a 2-by-1 column vector, it has the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `'SteerAngle'` is a scalar, it specifies the azimuth angle only. In this case, the elevation angle is assumed to be 0.

This option applies only when the `'SubarraySteering'` property of the System object is set to `'Phase'` or `'Time'`.

Example: `'SteerAngle',[20;30]`

Data Types: `double`

`ElementWeights` — Weights applied to elements within subarray
`1` (default) | complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array. Weights are applied to the individual elements within a subarray. Subarrays can have different dimensions and sizes.

If `ElementWeights` is a complex-valued N_{SE}-by-N matrix, N_{SE} is the number of elements in the largest subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray. Only the first K entries in each column are applied as weights where K is the number of elements in the corresponding subarray.

If `ElementWeights` is a 1-by-N cell array. Each cell contains a complex-valued column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this name-value pair, set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`D` — Directivity
M-by-L matrix

Directivity, returned as an M-by-L matrix. Each row corresponds to one of the M angles specified by `ANGLE`. Each column corresponds to one of the L frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Partitioned Array

Compute the directivity of a partitioned array formed from a single 20-element ULA with elements spaced one-quarter wavelength apart. The subarrays are then phase-steered towards 30 degrees azimuth. The directivities are computed at azimuth angles from 0 to 60 degrees.

```
c = physconst('LightSpeed');
fc = 3e8;
lambda = c/fc;
angsteer = [30;0];
ang = [0:10:60;0,0,0,0,0,0,0];
```

Create a partitioned ULA array using the `SubarraySelection` property.

```
myArray = phased.PartitionedArray('Array',...
    phased.ULA(20,lambda/4),'SubarraySelection',...
    [ones(1,10) zeros(1,10);zeros(1,10) ones(1,10)],...
    'SubarraySteering','Phase','PhaseShifterFrequency',fc);
```

Create the steering vector and compute the directivity.

```
myStv = phased.SteeringVector('SensorArray',myArray,...
    'PropagationSpeed',c);
d = directivity(myArray,fc,ang,'PropagationSpeed',c,'Weights',...
    step(myStv,fc,angsteer),'SteerAngle',angsteer)
```

```
d = 7×1

   -7.5778
   -4.7676
   -2.0211
   10.0996
    0.9714
   -3.5575
  -10.8439
```

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or

array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternAzimuth` | `patternElevation`

collectPlaneWave

System object: `phased.PartitionedArray`
Package: `phased`

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

`Y = collectPlaneWave(H,X,ANG)` returns the received signals at the sensor array, `H`, when the input signals indicated by `X` arrive at the array from the directions specified in `ANG`.

`Y = collectPlaneWave(H,X,ANG,FREQ)`, in addition, specifies the incoming signal carrier frequency in `FREQ`.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)`, in addition, specifies the signal propagation speed in `C`.

Input Arguments

`H`

Array object.

`X`

Incoming signals, specified as an M-column matrix. Each column of `X` represents an individual incoming signal.

`ANG`

Directions from which incoming signals arrive, in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column specifies the direction of arrival of the corresponding signal in `X`. Each column of `ANG` is in the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `ANG` is a row vector of length M, each entry in `ANG` specifies the azimuth angle. In this case, the corresponding elevation angle is assumed to be 0°.

`FREQ`

Carrier frequency of signal in hertz. `FREQ` must be a scalar.

Default: `3e8`

C

Propagation speed of signal in meters per second.

Default: Speed of light

Output Arguments

Y

Received signals. Y is an N-column matrix, where N is the number of subarrays in the array H. Each column of Y is the received signal at the corresponding subarray, with all incoming signals combined.

Examples

Plane Waves Received at Array Containing Subarrays

Simulate the received signal at a 16-element ULA partitioned into four 4-element ULAs.

Create a 16-element ULA, and partition it into 4-element ULAs.

```
ula = phased.ULA('NumElements',16);
array = phased.PartitionedArray('Array',ula,...
   'SubarraySelection',....
   [1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0;...
    0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0;...
    0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0;...
    0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1]);
```

Simulate received signals from 10° and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
sig = collectPlaneWave(array,randn(4,2),[10 30],1.0e8,physconst('LightSpeed'))
```

```
sig = 4×4 complex

  -0.0710 - 0.4765i   0.6616 - 0.4676i   0.6616 + 0.4676i  -0.0710 + 0.4765i
   2.1529 - 1.6304i   1.0607 + 0.4802i   1.0607 - 0.4802i   2.1529 + 1.6304i
  -0.6074 + 2.0037i  -2.3274 + 1.1797i  -2.3274 - 1.1797i  -0.6074 - 2.0037i
   0.0547 - 0.7644i   0.9768 - 0.6037i   0.9768 + 0.6037i   0.0547 + 0.7644i
```

Algorithms

`collectPlaneWave` modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. This method does not account for the response of individual elements in the array and only models the array factor among subarrays. Therefore, the result does not depend on whether the subarray is steered.

See Also
`uv2azel` | `phitheta2azel`

getElementPosition

System object: phased.PartitionedArray
Package: phased

Positions of array elements

Syntax

POS = getElementPosition(H)

Description

POS = getElementPosition(H) returns the element positions in the array H.

Input Arguments

H

Partitioned array object.

Output Arguments

POS

Element positions in array. POS is a 3-by-N matrix, where N is the number of elements in H. Each column of POS defines the position of an element in the local coordinate system, in meters, using the form [x; y; z].

Examples

Element Positions in Partitioned Array

Obtain the positions of the six elements in a partitioned array.

```
array = phased.PartitionedArray('Array',phased.URA('Size',[2 3]),...
    'SubarraySelection',[1 0 1 0 1 0; 0 1 0 1 0 1]);
pos = getElementPosition(array)
```

```
pos = 3×6

         0         0         0         0         0         0
   -0.5000   -0.5000         0         0    0.5000    0.5000
    0.2500   -0.2500    0.2500   -0.2500    0.2500   -0.2500
```

See Also

getSubarrayPosition

getNumElements

System object: `phased.PartitionedArray`
Package: `phased`

Number of elements in array

Syntax

`N = getNumElements(H)`

Description

`N = getNumElements(H)` returns the number of elements in the array object `H`.

Input Arguments

H

Partitioned array object.

Examples

Number of Elements in Partitioned Array

Obtain the number of elements in an array that is partitioned into subarrays.

```
array = phased.PartitionedArray('Array',phased.URA('Size',[2 3]),...
   'SubarraySelection',[1 0 1 0 1 0; 0 1 0 1 0 1]);
N = getNumElements(array)
```

```
N = 6
```

See Also

`getNumSubarrays`

getNumSubarrays

System object: phased.PartitionedArray
Package: phased

Number of subarrays in array

Syntax

N = getNumSubarrays(H)

Description

N = getNumSubarrays(H) returns the number of subarrays in the array object H. This number matches the number of rows in the SubarraySelection property of H.

Input Arguments

H

Partitioned array object.

Examples

Number of Subarrays in Partitioned Array

Obtain the number of subarrays in a partitioned array.

```
array = phased.PartitionedArray('Array',...
   phased.ULA('NumElements',5),...
   'SubarraySelection',[1 1 1 0 0; 0 0 1 1 1]);
N = getNumSubarrays(array)
```

```
N = 2
```

See Also

getNumElements

getSubarrayPosition

System object: `phased.PartitionedArray`
Package: `phased`

Positions of subarrays in array

Syntax

`POS = getSubarrayPosition(H)`

Description

`POS = getSubarrayPosition(H)` returns the subarray positions in the array `H`.

Input Arguments

H

Partitioned array object.

Output Arguments

POS

Subarrays positions in array. `POS` is a 3-by-N matrix, where N is the number of subarrays in `H`. Each column of `POS` defines the position of a subarray in the local coordinate system, in meters, using the form [x; y; z].

Examples

Subarray Positions in Partitioned Array

Obtain the positions of the two subarrays in a partitioned array.

```
array = phased.PartitionedArray('Array',phased.URA('Size',[2 3]),...
    'SubarraySelection',[1 0 1 0 1 0; 0 1 0 1 0 1]);
pos = getSubarrayPosition(array)
```

```
pos = 3×2

         0         0
         0         0
    0.2500   -0.2500
```

See Also

`getElementPosition`

isPolarizationCapable

System object: phased.PartitionedArray
Package: phased

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all its constituent sensor elements support polarization.

Input Arguments

h — Partitioned array

Partitioned array specified as a `phased.PartitionedArray` System object.

Output Arguments

flag — Polarization-capability flag

Polarization-capability flag returned as a Boolean value. This value is `true`, if the array supports polarization or `false`, if it does not.

Examples

Partitioned Array of Short-Dipole Antenna Elements Supports Polarization

Determine whether a partitioned array of `phased.ShortDipoleAntennaElements` supports polarization.

```
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[1e9 10e9]);
ulaarray = phased.ULA(4,'Element',antenna);
partitionedarray = phased.PartitionedArray('Array',ulaarray,...
     'SubarraySelection',[1 1 0 0; 0 0 1 1]);
isPolarizationCapable(partitionedarray)
```

```
ans = logical
   1
```

The returned value `1` shows that this array supports polarization.

pattern

System object: `phased.PartitionedArray`
Package: `phased`

Plot partitioned array directivity, field, and power patterns

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sArray,FREQ)` plots the 3-D array directivity pattern (in dBi) for the array specified in `sArray`. The operating frequency is specified in `FREQ`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`pattern(sArray,FREQ,AZ)` plots the array directivity pattern at the specified azimuth angle.

`pattern(sArray,FREQ,AZ,EL)` plots the array directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the array pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See “Convert plotResponse to pattern” on page 1-1169 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sArray` — Partitioned array
System object

Partitioned array, specified as a `phased.PartitionedArray` System object.

Example: `sArray= phased.PartitionedArray;`

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

CoordinateSystem — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Orientation` — Array orientation
`[0;0;0]`. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

`Normalize` — Display normalize pattern
`true` (default) | `false`

Display normalized pattern, specified as the comma-separated pair consisting of `'Normalize'` and a Boolean. Set this parameter to `true` to display a normalized pattern. This parameter does not apply when you set `'Type'` to `'directivity'`. Directivity patterns are already normalized.

Data Types: `logical`

`ShowArray` — View array geometry
`false` (default) | `true`

View the array geometry along with the 3D radiation pattern, specified as `false` or `true`.

Data Types: `logical`

`ShowLocalCoordinates` — Show local coordinate axes
`true` (default) | `false`

Show the local coordinate axes, specified as `true` or `false`.

Data Types: `logical`

`ShowColorbar` — Show colorbar
`true` (default) | `false`

Show the colorbar, specified as `true` or `false`.

Data Types: `logical`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Subarray weights
1 (default) | N-by-1 complex-valued column vector | N-by-L complex-valued matrix

Subarray weights, specified as the comma-separated pair consisting of `'Weights`' and an N-by-1 complex-valued column vector or N-by-M complex-valued matrix. The dimension N is the number of subarrays in the array. The dimension L is the number of frequencies specified by the `FREQ` argument.

Weights dimension	FREQ dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-L row vector	Applies a set of weights for the single frequency or for all L frequencies.
N-by-L complex-valued matrix	1-by-L row vector	Applies each of the L columns of `'Weights'` for the corresponding frequency in the `FREQ` argument.

Example: `'Weights',ones(N,M)`

Data Types: `double`

`SteerAngle` — Subarray steering angle
`[0;0]` (default) | scalar | 2-element column vector

Subarray steering angle, specified as the comma-separated pair consisting of `'SteerAngle'` and a scalar or a 2-by-1 column vector.

If `'SteerAngle'` is a 2-by-1 column vector, it has the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `'SteerAngle'` is a scalar, it specifies the azimuth angle only. In this case, the elevation angle is assumed to be 0.

This option applies only when the `'SubarraySteering'` property of the System object is set to `'Phase'` or `'Time'`.

Example: `'SteerAngle',[20;30]`

Data Types: `double`

`ElementWeights` — Weights applied to elements within subarray
`1` (default) | complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array. Weights are applied to the individual elements within a subarray. Subarrays can have different dimensions and sizes.

If `ElementWeights` is a complex-valued N_{SE}-by-N matrix, N_{SE} is the number of elements in the largest subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray. Only the first K entries in each column are applied as weights where K is the number of elements in the corresponding subarray.

If `ElementWeights` is a 1-by-N cell array. Each cell contains a complex-valued column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this name-value pair, set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

PAT — Array pattern
M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of PAT correspond to the dimensions of the output arguments AZ_ANG and EL_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

Examples

Azimuth Response of Partitioned ULA

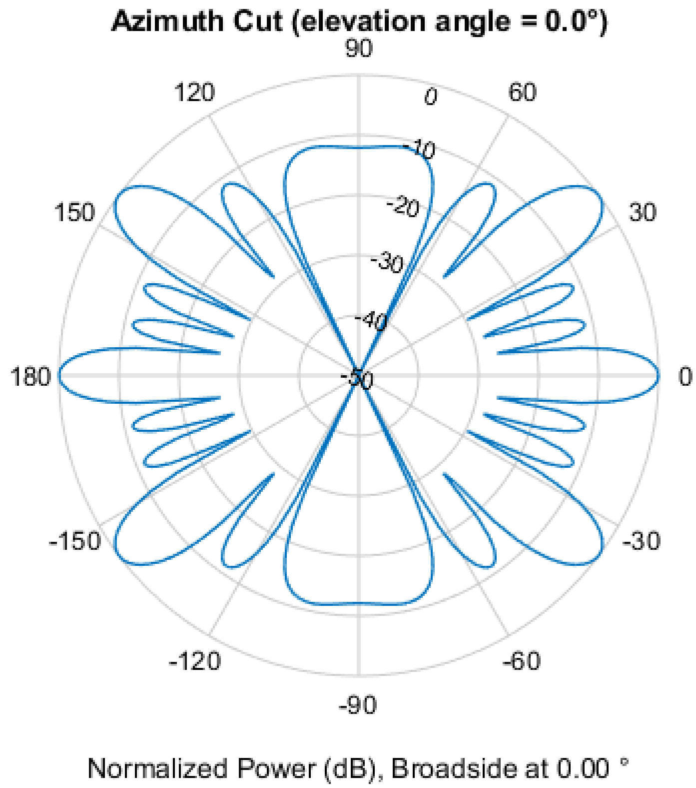
Plot the azimuth response of a 4-element ULA partitioned into two 2-element ULA's. The element spacing is one-half wavelength.

Create the ULA, and partition it into two 2-element ULA's.

```
sULA = phased.ULA('NumElements',4,'ElementSpacing',0.5);
sPA = phased.PartitionedArray('Array',sULA,...
    'SubarraySelection',[1 1 0 0;0 0 1 1]);
```

Plot the azimuth response of the array. Assume the operating frequency is 1 GHz and the propagation speed is the speed of light.

```
fc = 1e9;
pattern(sPA,fc,[-180:180],0,'Type','powerdb',...
    'CoordinateSystem','polar',...
    'Normalize',true)
```



Plot Pattern and Directivity of Partitioned URA Over Restricted Range of Angles

Convert a 2-by-6 URA of isotropic antenna elements into a 1-by-3 partitioned array so that each subarray of the partitioned array is a 2-by-2 URA. Assume that the frequency response of the elements lies between 1 and 6 GHz. The elements are spaced one-half wavelength apart corresponding to the highest frequency of the element response. Plot an azimuth cut from -50 to 50 degrees for different two sets of weights. For partitioned arrays, weights are applied to the subarrays instead of the elements.

Create partitioned array

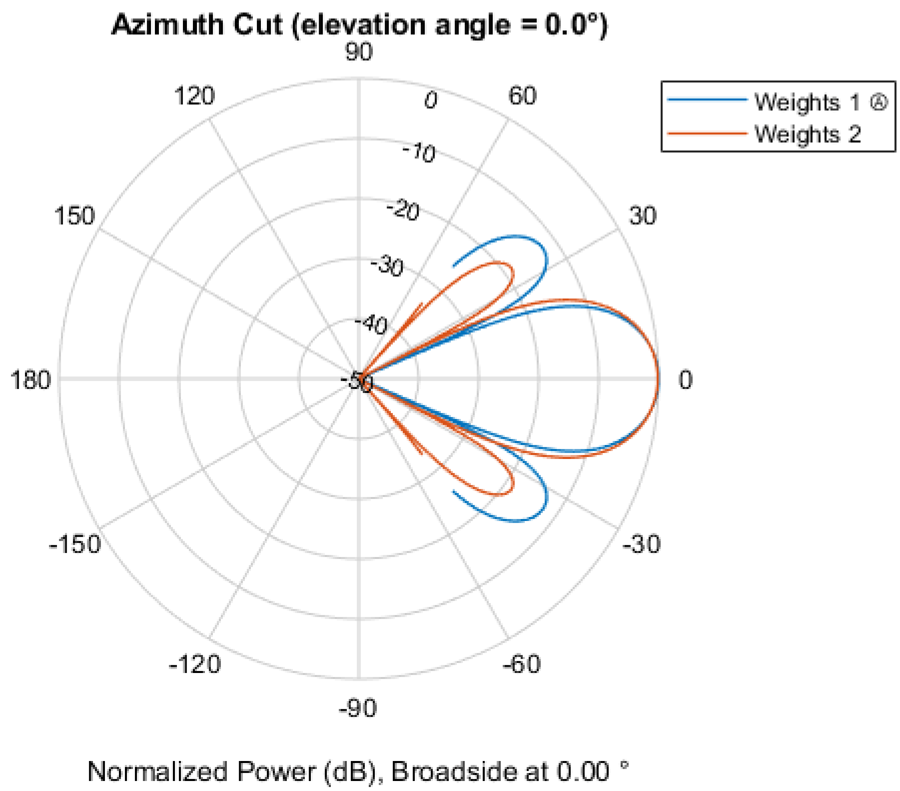
```
fmin = 1e9;
fmax = 6e9;
c = physconst('LightSpeed');
lam = c/fmax;
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[fmin,fmax],...
    'BackBaffled',false);
sURA = phased.URA('Element',sIso,'Size',[2,6],...
    'ElementSpacing',[lam/2,lam/2]);
subarraymap = [[1,1,1,1,0,0,0,0,0,0,0,0];...
    [0,0,0,0,1,1,1,1,0,0,0,0];...
    [0,0,0,0,0,0,0,0,1,1,1,1]];
```

```
sPA = phased.PartitionedArray('Array',sURA,...
    'SubarraySelection',subarraymap);
```

Plot power pattern

Plot the response of the array at 5 GHz over the restricted range of azimuth angles.

```
fc = 5e9;
wts = [[1,1,1]',[.862,1.23,.862]'];
pattern(sPA,fc,[-50:0.1:50],0,...
    'Type','powerdb',...
    'CoordinateSystem','polar',...
    'Weights',wts)
```

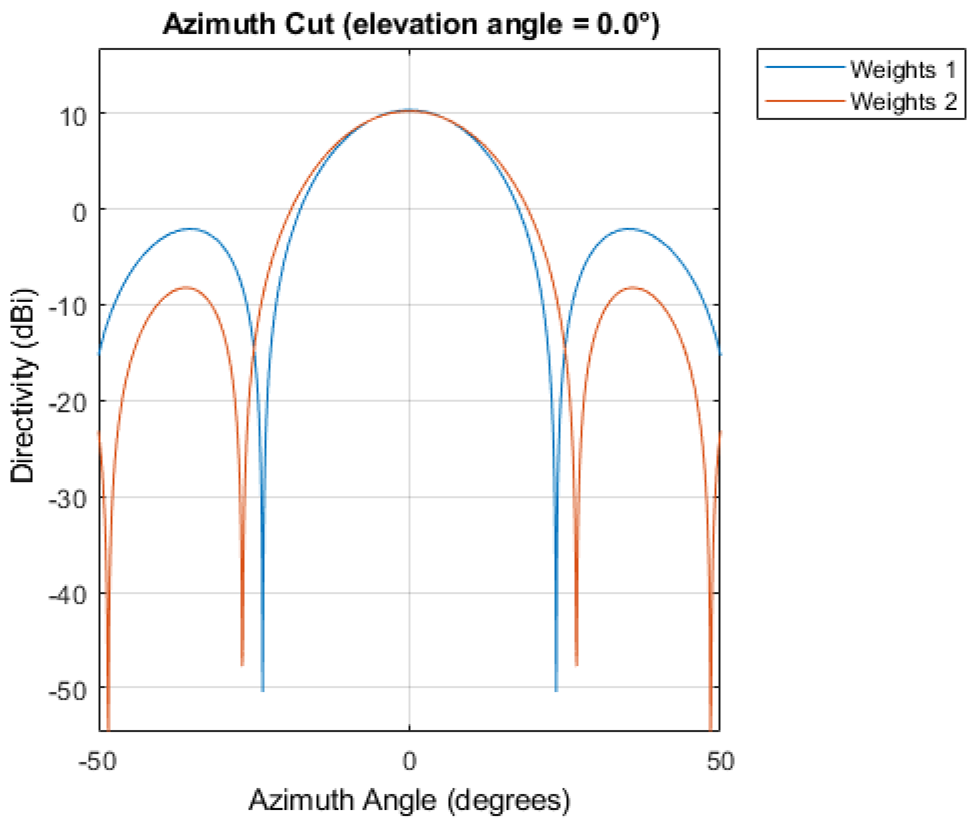


The plot of the response shows the broadening of the main lobe and the reduction of the strength of the sidelobes caused by the weight tapering.

Plot directivity

Plot an azimuth cut of the directivity of the array at 5 GHz over the restricted range of azimuth angles for the two different sets of weights.

```
fc = 5e9;
wts = [[1,1,1]',[.862,1.23,.862]'];
pattern(sPA,fc,[-50:0.1:50],0,...
    'Type','directivity',...
    'CoordinateSystem','rectangular',...
    'Weights',wts)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td><code>'Format'</code> and <code>'RespCut'</code> name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using <code>plotResponse</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'RespCut'</code> to <code>'Az'</code> or <code>'El'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using either the <code>'AzimuthAngles'</code> or <code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'RespCut'</code> to <code>'3D'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using both the <code>'AzimuthAngles'</code> and<code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'RespCut'</code> to<code>'U'</code>. Set <code>'Format'</code> to <code>'UV'</code>. Set the display range using the <code>'UGrid'</code> name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'RespCut'</code> to<code>'3D'</code>. Set <code>'Format'</code> to</td></tr>
</table>
</td>
<td><code>'CoordinateSystem'</code> name-value pair used together with the <code>AZ</code> and <code>EL</code> input arguments.

<code>'CoordinateSystem'</code> has the same options as the <code>plotResponse</code> method <code>'Format'</code>name-value pair, except that <code>'line'</code> is now named <code>'rectangular'</code>. The table shows how to create different types of plots using <code>pattern</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify either <code>AZ</code> or <code>EL</code> as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify both <code>AZ</code> and <code>EL</code> as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space vector.</td></tr>
</table>
If you set <code>CoordinateSystem</code> to <code>'uv'</code>, enter the UV grid values using <code>AZ</code> and <code>EL</code>.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `patternElevation`

patternAzimuth

System object: phased.PartitionedArray
Package: phased

Plot partitioned array directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sArray,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(sArray,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sArray,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

sArray — Partitioned array

System object

Partitioned array, specified as a `phased.PartitionedArray` System object.

Example: `sArray= phased.PartitionedArray;`

FREQ — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Subarray weights
M-by-1 complex-valued column vector

Subarray weights, specified as the comma-separated pair consisting of `'Weights'` and an M-by-1 complex-valued column vector. Subarray weights are applied to the subarrays of the array to produce array steering, tapering, or both. The dimension M is the number of subarrays in the array.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

SteerAngle — Subarray steering angle
`[0;0]` (default) | scalar | 2-element column vector

Subarray steering angle, specified as the comma-separated pair consisting of `'SteerAngle'` and a scalar or a 2-by-1 column vector.

If `'SteerAngle'` is a 2-by-1 column vector, it has the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `'SteerAngle'` is a scalar, it specifies the azimuth angle only. In this case, the elevation angle is assumed to be 0.

This option applies only when the `'SubarraySteering'` property of the System object is set to `'Phase'` or `'Time'`.

Example: `'SteerAngle',[20;30]`

Data Types: `double`

ElementWeights — Weights applied to elements within subarray
`1` (default) | complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array. Weights are applied to the individual elements within a subarray. Subarrays can have different dimensions and sizes.

If `ElementWeights` is a complex-valued N_{SE}-by-N matrix, N_{SE} is the number of elements in the largest subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray. Only the first K entries in each column are applied as weights where K is the number of elements in the corresponding subarray.

If `ElementWeights` is a 1-by-N cell array. Each cell contains a complex-valued column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this name-value pair, set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

Azimuth — Azimuth angles
`[-180:180]` (default) | 1-by-P real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

Examples

Plot Azimuth Directivity of Partitioned URA

Convert a 2-by-6 URA of isotropic antenna elements into a 1-by-3 partitioned array so that each subarray of the partitioned array is a 2-by-2 URA. Assume that the frequency response of the elements lies between 1 and 6 GHz. The elements are spaced one-half wavelength apart corresponding to the highest frequency of the element response. Plot the azimuth directivity. For partitioned arrays, weights are applied to the subarrays instead of the elements.

Create partitioned array

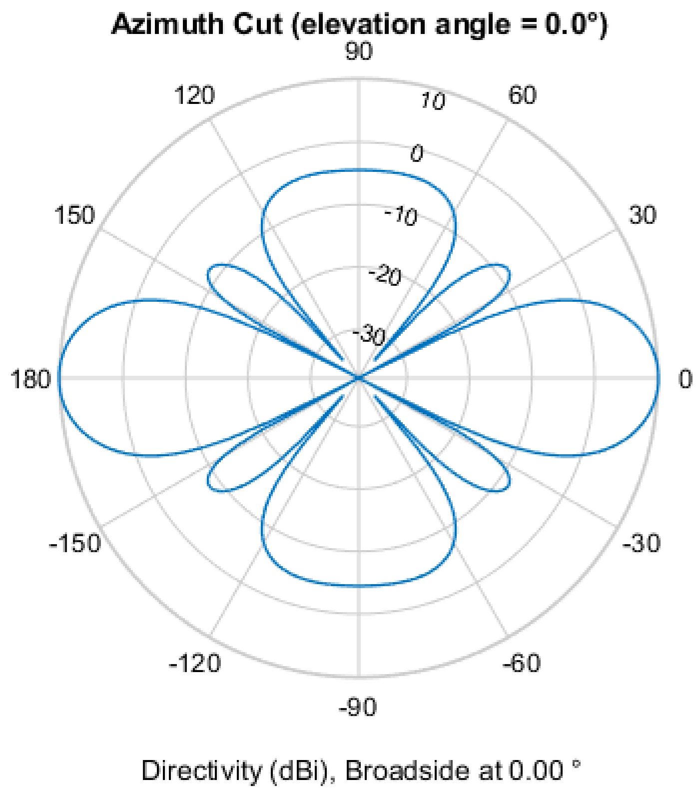
```
fmin = 1e9;
fmax = 6e9;
c = physconst('LightSpeed');
lam = c/fmax;
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[fmin,fmax],...
    'BackBaffled',false);
sURA = phased.URA('Element',sIso,'Size',[2,6],...
    'ElementSpacing',[lam/2,lam/2]);
subarraymap = [[1,1,1,1,0,0,0,0,0,0,0,0];...
    [0,0,0,0,1,1,1,1,0,0,0,0];...
    [0,0,0,0,0,0,0,0,1,1,1,1]];
sPA = phased.PartitionedArray('Array',sURA,...
    'SubarraySelection',subarraymap);
```

Plot azimuth directivity pattern

Plot the response of the array at 5 GHz

```
fc = 5e9;
wts = [0.862,1.23,0.862]';
patternAzimuth(sPA,fc,0,...
```

```
    'Type','directivity',...
    'PropagationSpeed',physconst('LightSpeed'),...
    'Weights',wts)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: phased.PartitionedArray
Package: phased

Plot partitioned array directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

patternElevation(sArray,FREQ) plots the 2-D array directivity pattern versus elevation (in dBi) for the array sArray at zero degrees azimuth angle. When AZ is a vector, multiple overlaid plots are created. The argument FREQ specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

patternElevation(sArray,FREQ,AZ), in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by AZ. When AZ is a vector, multiple overlaid plots are created.

patternElevation(sArray,FREQ,AZ,Name,Value) plots the array pattern with additional options specified by one or more Name,Value pair arguments.

PAT = patternElevation(___) returns the array pattern. PAT is a matrix whose entries represent the pattern at corresponding sampling points specified by the 'Elevation' parameter and the AZ input argument.

Input Arguments

sArray — Partitioned array
System object

Partitioned array, specified as a phased.PartitionedArray System object.

Example: sArray= phased.PartitionedArray;

FREQ — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Subarray weights
M-by-1 complex-valued column vector

Subarray weights, specified as the comma-separated pair consisting of 'Weights' and an M-by-1 complex-valued column vector. Subarray weights are applied to the subarrays of the array to produce array steering, tapering, or both. The dimension M is the number of subarrays in the array.

Example: 'Weights',ones(10,1)

Data Types: double
Complex Number Support: Yes

SteerAngle — Subarray steering angle
[0;0] (default) | scalar | 2-element column vector

Subarray steering angle, specified as the comma-separated pair consisting of 'SteerAngle' and a scalar or a 2-by-1 column vector.

If 'SteerAngle' is a 2-by-1 column vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If 'SteerAngle' is a scalar, it specifies the azimuth angle only. In this case, the elevation angle is assumed to be 0.

This option applies only when the 'SubarraySteering' property of the System object is set to 'Phase' or 'Time'.

Example: 'SteerAngle',[20;30]

Data Types: double

ElementWeights — Weights applied to elements within subarray
1 (default) | complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array. Weights are applied to the individual elements within a subarray. Subarrays can have different dimensions and sizes.

If ElementWeights is a complex-valued N_{SE}-by-N matrix, N_{SE} is the number of elements in the largest subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray. Only the first K entries in each column are applied as weights where K is the number of elements in the corresponding subarray.

If ElementWeights is a 1-by-N cell array. Each cell contains a complex-valued column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this name-value pair, set the SubarraySteering property of the array to 'Custom'.

Data Types: double
Complex Number Support: Yes

Elevation — Elevation angles
[-90:90] (default) | 1-by-P real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Plot Elevation Directivity of Partitioned URA

Convert a 2-by-6 URA of isotropic antenna elements into a 1-by-3 partitioned array so that each subarray of the partitioned array is a 2-by-2 URA. Assume that the frequency response of the elements lies between 1 and 6 GHz. The elements are spaced one-half wavelength apart corresponding to the highest frequency of the element response. Plot the directivity for elevation angles from -45 to 45 degrees. For partitioned arrays, weights are applied to the subarrays instead of the elements.

Create partitioned array

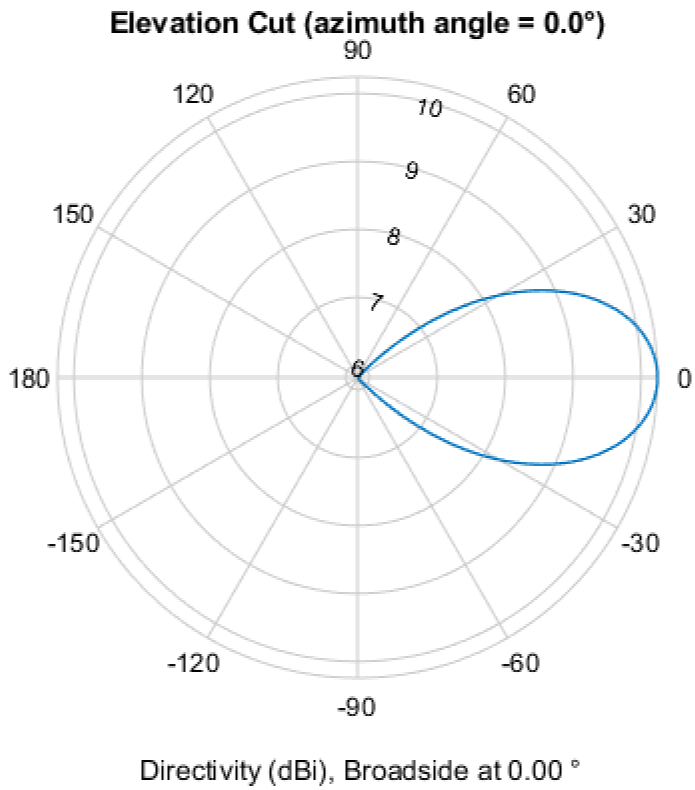
```
fmin = 1e9;
fmax = 6e9;
c = physconst('LightSpeed');
lam = c/fmax;
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[fmin,fmax],...
    'BackBaffled',false);
sURA = phased.URA('Element',sIso,'Size',[2,6],...
    'ElementSpacing',[lam/2,lam/2]);
subarraymap = [[1,1,1,1,0,0,0,0,0,0,0,0];...
    [0,0,0,0,1,1,1,1,0,0,0,0];...
    [0,0,0,0,0,0,0,0,1,1,1,1]];
sPA = phased.PartitionedArray('Array',sURA,...
    'SubarraySelection',subarraymap);
```

Plot elevation directivity pattern

Plot the response of the array at 5 GHz

```
fc = 5e9;
wts = [0.862,1.23,0.862]';
```

```
azimangle = 0;
patternElevation(sPA,fc,azimangle,...
    'Type','directivity',...
    'PropagationSpeed',physconst('LightSpeed'),...
    'Elevation',[-45:45],...
    'Weights',wts)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or

array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth`

plotResponse

System object: phased.PartitionedArray
Package: phased

Plot response pattern of array

Syntax

```
plotResponse(H,FREQ,V)
plotResponse(H,FREQ,V,Name,Value)
hPlot = plotResponse( ___ )
```

Description

plotResponse(H,FREQ,V) plots the array response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in FREQ. The propagation speed is specified in V.

plotResponse(H,FREQ,V,Name,Value) plots the array response with additional options specified by one or more Name,Value pair arguments.

hPlot = plotResponse(___) returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Array object.

FREQ

Operating frequency in hertz. Typical values are within the range specified by a property of H.Array.Element. That property is named FrequencyRange or FrequencyVector, depending on the type of element in the array. The element has zero response at frequencies outside that range. If FREQ is a nonscalar row vector, the plot shows multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

CutAngle

Cut angle specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

Format

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

NormalizeResponse

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

OverlayFreq

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, then `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and `RespCut` is not `'3D'`.

Default: `true`

Polarization

Specify the polarization options for plotting the array response pattern. The allowable values are `|'None' | 'Combined' | 'H' | 'V' |` where:

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

RespCut

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.

- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

`SteerAng`

Subarray steering angle. `SteerAng` can be either a 2-element column vector or a scalar.

If `SteerAng` is a 2-element column vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If `SteerAng` is a scalar, it specifies the azimuth angle. In this case, the elevation angle is assumed to be 0.

This option is applicable only if the `SubarraySteering` property of `H` is `'Phase'` or `'Time'`.

Default: `[0;0]`

`Unit`

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

`Weights`

Weight values applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. The dimension *N* is the number of subarrays in the array. The interpretation of *M* depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

AzimuthAngles

Azimuth angles for plotting subarray response, specified as a row vector. The AzimuthAngles parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'Az' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the AzimuthAngles and ElevationAngles parameters simultaneously.

Default: [-180:180]

ElevationAngles

Elevation angles for plotting subarray response, specified as a row vector. The ElevationAngles parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'El' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the ElevationAngles and AzimuthAngles parameters simultaneously.

Default: [-90:90]

UGrid

U coordinate values for plotting subarray response, specified as a row vector. The UGrid parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to 'U' or '3D'. The values of UGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the UGrid and VGrid parameters simultaneously.

Default: [-1:0.01:1]

VGrid

V coordinate values for plotting subarray response, specified as a row vector. The VGrid parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to '3D'. The values of VGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the VGrid and UGrid parameters simultaneously.

Default: [-1:0.01:1]

Examples

Azimuth Response of Partitioned ULA

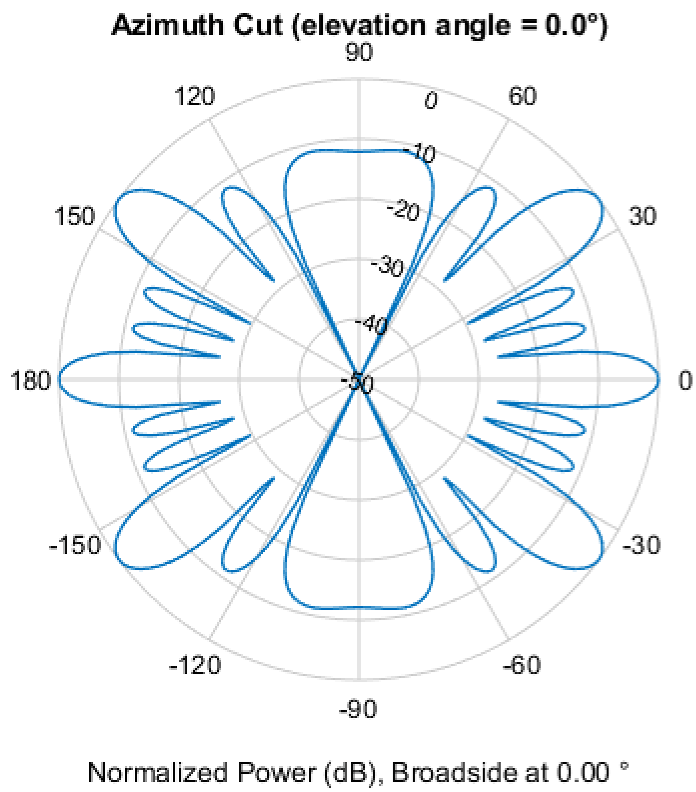
Plot the azimuth response of a 4-element ULA partitioned into two 2-element ULA's. The element spacing is one-half wavelength.

Create the ULA, and partition it into two 2-element ULA's.

```
sULA = phased.ULA('NumElements',4,'ElementSpacing',0.5);
sPA = phased.PartitionedArray('Array',sULA,...
    'SubarraySelection',[1 1 0 0;0 0 1 1]);
```

Plot the azimuth response of the array. Assume the operating frequency is 1 GHz and the propagation speed is the speed of light.

```
fc = 1e9;
pattern(sPA,fc,[-180:180],0,'Type','powerdb',...
    'CoordinateSystem','polar',...
    'Normalize',true)
```



Plot Response and Directivity of Partitioned URA over Restricted Range of Angles

Convert a 2-by-6 URA of isotropic antenna elements into a 1-by-3 partitioned array so that each subarray of the partitioned array is a 2-by-2 URA. Assume that the frequency response of the elements lies between 1 and 6 GHz. The elements are spaced one-half wavelength apart corresponding to the highest frequency of the element response. Plot an azimuth cut from -50 to 50 degrees for two different sets of weights. For partitioned arrays, weights are applied to the subarrays instead of the elements.

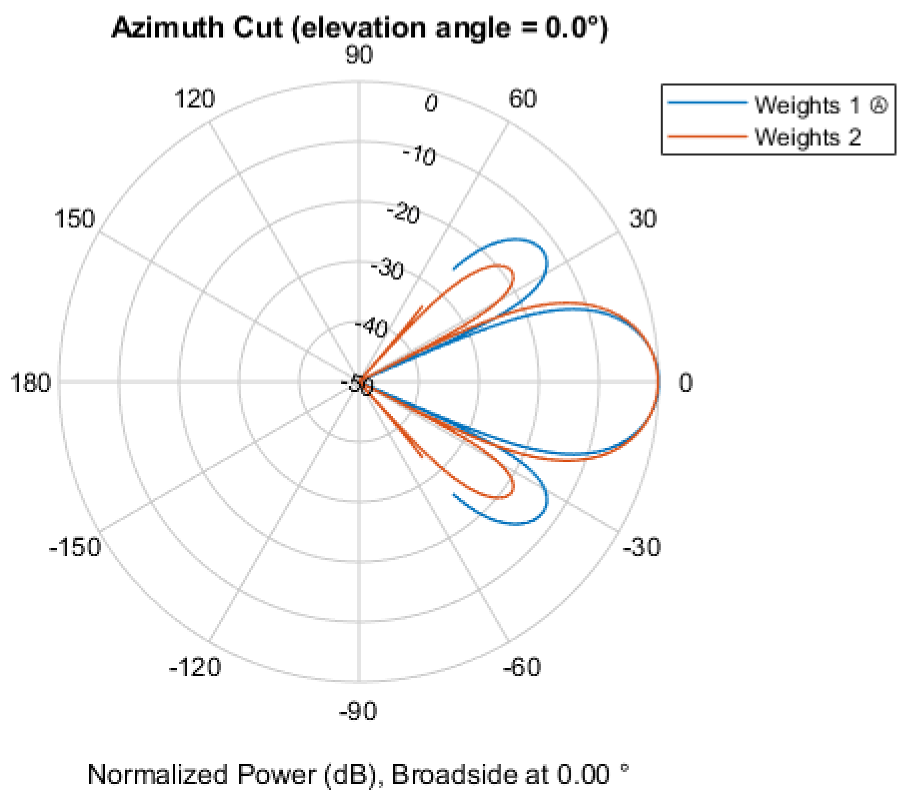
Set up the partitioned array.

```
fmin = 1e9;
fmax = 6e9;
c = physconst('LightSpeed');
lam = c/fmax;
s_iso = phased.IsotropicAntennaElement(...
```

```
    'FrequencyRange',[fmin,fmax],...
    'BackBaffled',false);
s_ura = phased.URA('Element',s_iso,'Size',[2,6],...
    'ElementSpacing',[lam/2,lam/2]);
subarraymap = [[1,1,1,1,0,0,0,0,0,0,0,0];...
    [0,0,0,0,1,1,1,1,0,0,0,0];...
    [0,0,0,0,0,0,0,0,1,1,1,1]];
s_pa = phased.PartitionedArray('Array',s_ura,...
    'SubarraySelection',subarraymap);
```

Plot the response of the array at 5 GHz over the restricted range of azimuth angles.

```
fc = 5e9;
wts = [[1,1,1]',[.862,1.23,.862]'];
plotResponse(s_pa,fc,c,'RespCut','Az',...
    'AzimuthAngles',[-50:0.1:50],...
    'Unit','db','Format','Polar',...
    'Weights',wts);
```

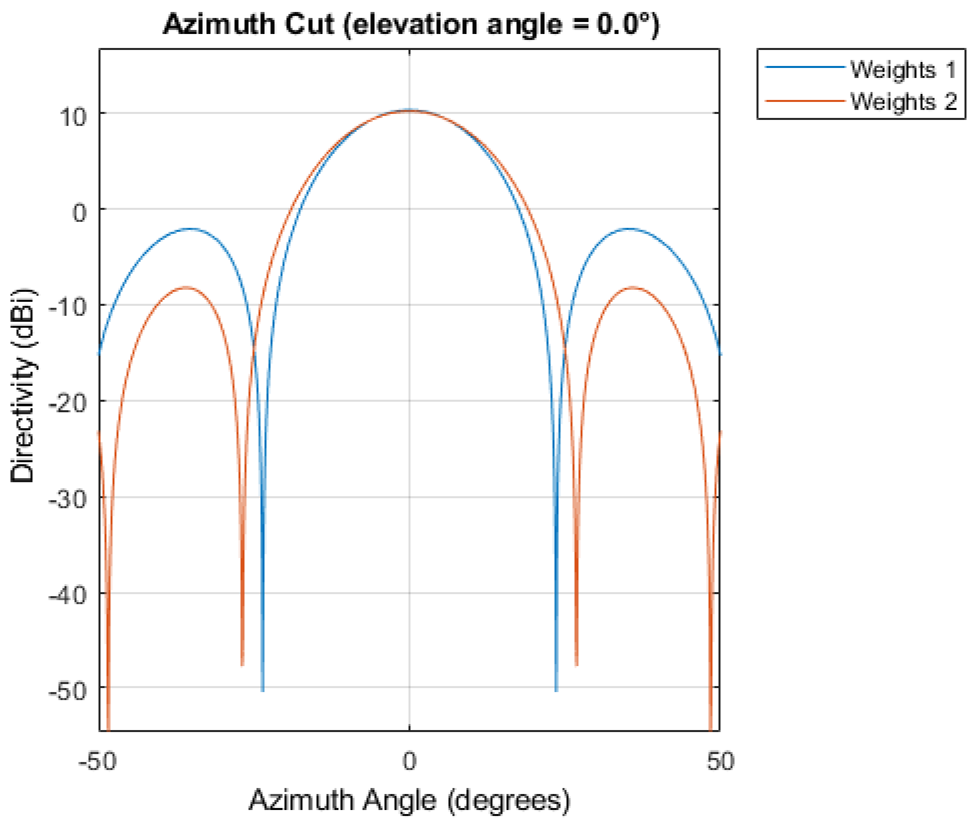


The plot of the response shows the broadening of the main lobe and the reduction of the strength of the sidelobes caused by the weight tapering.

Next, plot an azimuth cut of the directivity of the array at 5 GHz over the restricted range of azimuth angles for the two different sets of weights.

```
fc = 5e9;
wts = [[1,1,1]',[.862,1.23,.862]'];
plotResponse(s_pa,fc,c,'RespCut','Az',...
```

```
'AzimuthAngles',[-50:0.1:50],...
'Unit','dbi',...
'Weights',wts);
```



See Also

`uv2azel` | `azel2uv`

step

System object: `phased.PartitionedArray`
Package: `phased`

Output responses of subarrays

Syntax

```
RESP = step(H,FREQ,ANG,V)
RESP = step(H,FREQ,ANG,V,STEERANGLE)
RESP = step(H,FREQ,ANG,V,WS)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG,V)` returns the responses `RESP` of the subarrays in the array, at operating frequencies specified in `FREQ` and directions specified in `ANG`. The phase center of each subarray is at its geometric center. `V` is the propagation speed. The elements within each subarray are connected to the subarray phase center using an equal-path feed.

`RESP = step(H,FREQ,ANG,V,STEERANGLE)` uses `STEERANGLE` as the steering direction of the subarray. This syntax is available when you set the `SubarraySteering` property to either `'Phase'` or `'Time'`.

`RESP = step(H,FREQ,ANG,V,WS)` uses `WS` as the subarray element weights. This syntax is available when you set the `SubarraySteering` property to `'Custom'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Partitioned array object.

FREQ

Operating frequencies of array in hertz. `FREQ` is a row vector of length L. Typical values are within the range specified by a property of `H.Array.Element`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has zero response at frequencies outside that range.

ANG

Directions in degrees. ANG can be either a 2-by-M matrix or a row vector of length M.

If ANG is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If ANG is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

V

Propagation speed in meters per second. This value must be a scalar.

STEERANGLE

Subarray steering direction. STEERANGLE can be either a 2-element column vector or a scalar.

If STEERANGLE is a 2-element column vector, it has the form `[azimuth; elevation]`. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If STEERANGLE is a scalar, it specifies the direction's azimuth angle. In this case, the elevation angle is assumed to be 0.

WS

Subarray element weights

Subarray element weights, specified as a complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays.

Subarrays do not have to have the same dimensions and sizes. In this case, you specify subarray weights as

- an N_{SE}-by-N matrix, where N_{SE} is the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray.
- a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `SubarraySteering` to `'Custom'`.

Output Arguments

RESP

Voltage responses of the subarrays of a phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, RESP, has the dimensions N-by-M-by-L. The size N represents the number of subarrays in the phased array, M represents the number of angles specified in ANG, and L represents the number of frequencies specified in FREQ. For a particular subarray, each column of RESP contains the responses of the subarray for the corresponding direction specified in ANG. Each of the L pages of RESP contains the responses of the subarrays for the corresponding frequency specified in FREQ.
- If the array is capable of supporting polarization, the voltage response, RESP, is a MATLAB struct containing two fields, RESP.H and RESP.V. The field RESP.H represents the array's horizontal polarization response while RESP.V represents the array's vertical polarization response. Each field has the dimensions N-by-M-by-L. The size N represents the number of subarrays in the phased array, M represents the number of angles specified in ANG, and L represents the number of frequencies specified in FREQ. For a particular subarray, each column of RESP contains the responses of the subarray for the corresponding direction specified in ANG. Each of the L pages of RESP contains the responses of the subarrays for the corresponding frequency specified in FREQ.

Examples

Response of Subarrays in Partitioned ULA

Calculate the response at boresight of a 4-element ULA partitioned into two 2-element ULAs.

Set up the partitioned array.

```
hula = phased.ULA('NumElements',4,'ElementSpacing',0.5);
partitionedarray = phased.PartitionedArray('Array',hula,...
   'SubarraySelection',[1 1 0 0;0 0 1 1]);
```

Calculate the response of the subarrays at boresight. Assume the operating frequency is 1 GHz and the propagation speed is the speed of light.

```
resp = partitionedarray(1.0e9,[0;0],physconst('Lightspeed'))
```

```
resp = 2×1

     2
     2
```

Subarray Element Weights for Partitioned Array

Create a partitioned URA array with three subarrays of different sizes. The subarrays have 8, 16, and 32 elements. Use different sets of subarray element weights for each subarray.

Create a 4-by-56 element URA.

```
antenna = phased.IsotropicAntennaElement;
fc = 300e6;
c = physconst('LightSpeed');
lambda = c/fc;
n1 = 2^3;
```

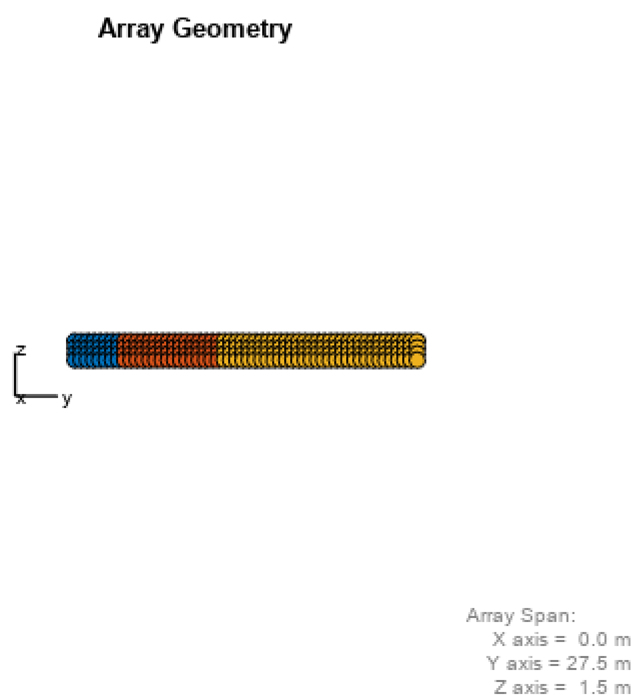
```
n2 = 2^4;
n3 = 2^5;
nrows = 4;
ncols = n1 + n2 + n3;
array = phased.URA('Element',antenna,'Size',[nrows,ncols]);
```

Select the three subarrays by setting the selection matrix.

```
sel1 = zeros(nrows,ncols);
sel2 = sel1;
sel3 = sel1;
sel = zeros(3,nrows*ncols);
for r = 1:nrows
    sel1(r,1:n1) = 1;
    sel2(r,(n1+1):(n1+n2)) = 1;
    sel3(r,((n1+n2)+1):ncols) = 1;
end
sel(1,:) = sel1(:);
sel(2,:) = sel2(:);
sel(3,:) = sel3(:);
```

Create the partitioned array.

```
partarray = phased.PartitionedArray('Array',array, ...
    'SubarraySelection',sel,'SubarraySteering','Custom');
viewArray(partarray,'ShowSubarray','All');
```



Set weights for each subarray and get the response of each subarray. Put the weights in a cell array.

```
wts1 = ones(nrows*n1,1);
wts2 = 1.5*ones(nrows*n2,1);
wts3 = 3*ones(nrows*n3,1);
resp = partarray(fc,[30;0],c,{wts1,wts2,wts3})
```

```
resp = 3×1 complex

   0.0246 + 0.0000i
   0.0738 - 0.0000i
   0.2951 - 0.0000i
```

See Also

uv2azel | phitheta2azel

viewArray

System object: phased.PartitionedArray
Package: phased

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

viewArray(H) plots the geometry of the array specified in H.

viewArray(H,Name,Value) plots the geometry of the array, with additional options specified by one or more Name,Value pair arguments.

hPlot = viewArray(___) returns the handles of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

H

Array object.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Parent

Handle to the axes along which the array geometry is displayed.

ShowNormals

Set this value to true to show the normal directions of all elements of the array. Set this value to false to plot the elements without showing normal directions.

Default: false

ShowLocalCoordinates

Logical flag specifying whether to show the local coordinate axes.

Default: `true`

ShowAnnotation

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Default: `true`

Orientation

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Default: `[0;0;0]`

ShowTaper

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color. The default value is `false`.

Default: `false`

ShowIndex

Vector specifying the element indices to show in the figure. Each number in the vector must be an integer between 1 and the number of elements. You can also specify the value `'All'` to show indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

ShowSubarray

Vector specifying the indices of subarrays to highlight in the figure. Each number in the vector must be an integer between 1 and the number of subarrays. You can also specify the value `'All'` to highlight all subarrays of the array or `'None'` to suppress the subarray highlighting. The highlighting uses different colors for different subarrays, and white for elements that occur in multiple subarrays.

Default: `'All'`

Title

Character vector specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

Handles of array elements in figure window.

Examples

Highlight Overlapped Subarrays

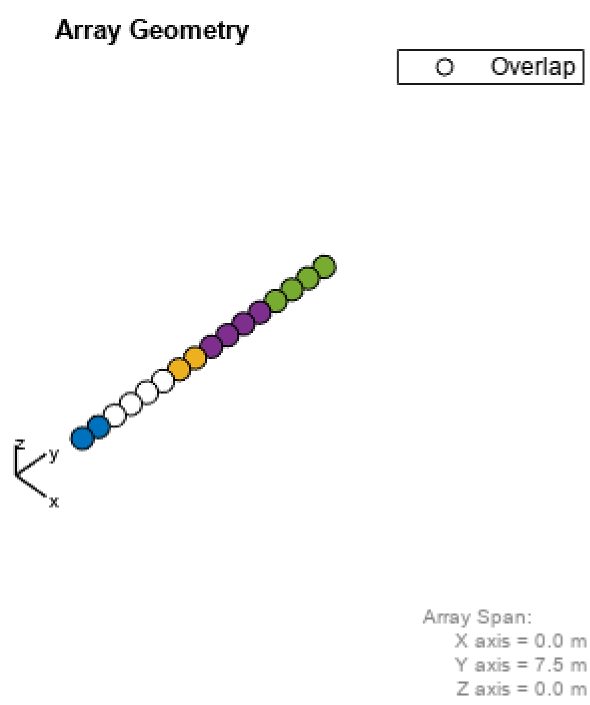
Display the geometry of a uniform linear array having overlapped subarrays.

Create a 16-element ULA that has five 4-element subarrays. Some elements occur in more than one subarray.

```
h = phased.ULA(16);
ha = phased.PartitionedArray('Array',h,...
    'SubarraySelection',...
    [1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0;...
    0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0;...
    0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0;...
    0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0;...
    0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1]);
```

Display the geometry of the array, highlighting all subarrays.

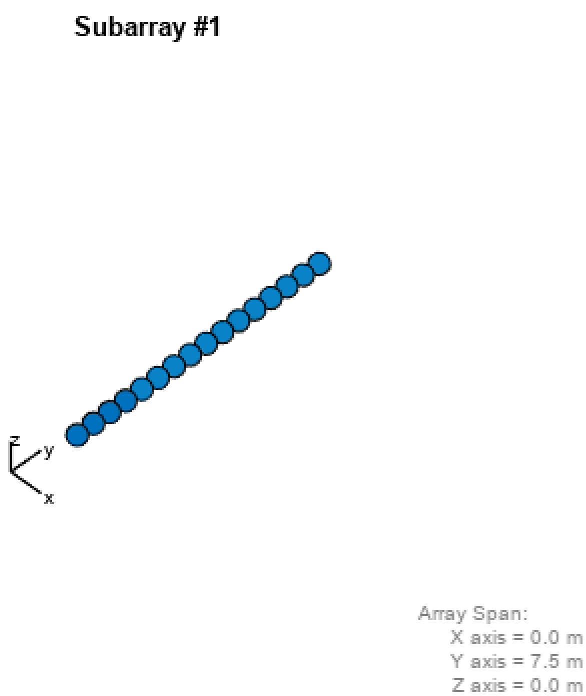
```
viewArray(ha);
```

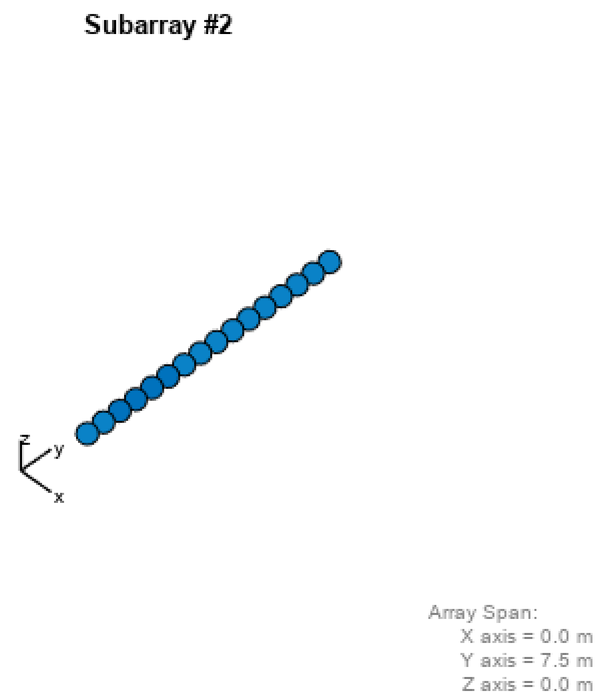


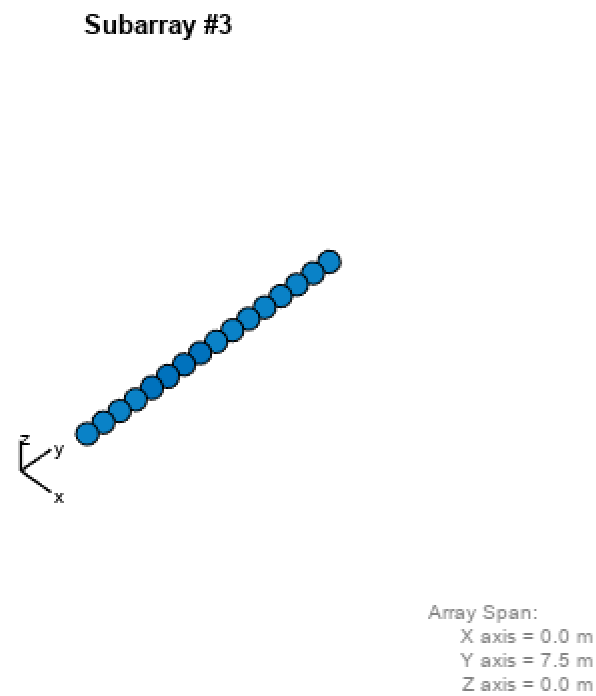
Each color other than white represents a different subarray. White represents elements that occur in multiple subarrays.

Examine the overlapped subarrays by creating separate figures that highlight the first, second, and third subarrays. In each figure, dark blue represents the highlighted elements.

```
for idx = 1:3
    figure;
    viewArray(ha,'ShowSubarray',idx,...
        'Title',['Subarray #' num2str(idx)]);
end
```




Subarray #2
z
y
x
Array Span:
X axis = 0.0 m
Y axis = 7.5 m
Z axis = 0.0 m



See Also
phased.ArrayResponse

Topics
“Phased Array Gallery”

phased.PhaseCodedWaveform

Package: phased

Phase-coded pulse waveform

Description

The PhaseCodedWaveform object creates a phase-coded pulse waveform.

To obtain waveform samples:

1. Define and set up your phase-coded pulse waveform. See "Construction" on page 1-1204.
2. Call step to generate the phase-coded pulse waveform samples according to the properties of phased.PhaseCodedWaveform. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations. When the only argument to the step method is the System object itself, replace y = step(obj) by y = obj().

Construction

H = phased.PhaseCodedWaveform creates a phase-coded pulse waveform System object, H. The object generates samples of a phase-coded pulse.

H = phased.PhaseCodedWaveform(Name,Value) creates a phase-coded pulse waveform object, H, with additional options specified by one or more Name,Value pair arguments. Name is a property name on page 1-1204, and Value is the corresponding value. Name must appear inside single quotes (''). You can specify several name-value pair arguments in any order as Name1,Value1, …,NameN,ValueN.

Properties

SampleRate

Sample rate

Specify the sample rate in hertz as a positive scalar. The default value of this property corresponds to 1 MHz. The value of this property must satisfy these constraints:

- (SampleRate./PRF) is a scalar or vector that contains only integers — the number of samples in a pulse must be an integer.
- (SampleRate*ChipWidth) is an integer value — the number of samples in a chip must be an integer.

Default: 1e6

`Code`

Phase code type

Specify the phase code type used in phase modulation. Valid values are:

- `'Barker'`
- `'Frank'`
- `'P1'`
- `'P2'`
- `'P3'`
- `'P4'`
- `'Px'`
- `'Zadoff-Chu'`

Default: `'Frank'`

`ChipWidth`

Time duration of each chip

Specify the time duration of each chip in a phase-coded waveform as a positive scalar. Units are seconds. For this waveform, the pulse duration is equal to the product of the chip width and number of chips.

The value of this property must satisfy these constraints:

- `ChipWidth` is less than or equal to `(1./(NumChips*PRF))` — the total time duration of all chips cannot exceed the duration of the pulse.
- `(SampleRate*ChipWidth)` is an integer value — the number of samples in a chip must be an integer.

Default: `1e-5`

`NumChips`

Number of chips

Specify the number of chips per pulse in a phase-coded waveform as a positive integer. The value of this property must be less than or equal to `(1./(ChipWidth*PRF))` — the total time duration of all chips cannot exceed the pulse repetition interval.

The table shows additional constraints on the number of chips for different code types.

If the `Code` property is ...	Then the `NumChips` property must be...
`'Frank'`, `'P1'`, or `'Px'`	A perfect square
`'P2'`	An even number that is a perfect square
`'Barker'`	2, 3, 4, 5, 7, 11, or 13

Default: 4

SequenceIndex

Zadoff-Chu sequence index

Specify the sequence index used in Zadoff-Chu code as a positive integer. This property applies only when you set the Code property to 'Zadoff-Chu'. The value of SequenceIndex must be relatively prime to the value of the NumChips property.

Default: 1

PRF

Pulse repetition frequency

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The*PRF* must satisfy these restrictions:

- The product of *PRF* and *PulseWidth* must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of PRF must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* using property settings alone or using property settings in conjunction with the prfidx input argument of the step method.

- When PRFSelectionInputPort is false, you set the *PRF* using properties only. You can
  - implement a constant *PRF* by specifying PRF as a positive real-valued scalar.
  - implement a staggered *PRF* by specifying PRF as a row vector with positive real-valued entries. Then, each call to the step method uses successive elements of this vector for the *PRF*. If the last element of the vector is reached, the process continues cyclically with the first element of the vector.
- When PRFSelectionInputPort is true, you can implement a selectable *PRF* by specifying PRF as a row vector with positive real-valued entries. But this time, when you execute the step method, select a *PRF* by passing an argument specifying an index into the *PRF* vector.

In all cases, the number of output samples is fixed when you set the OutputFormat property to 'Samples'. When you use a varying *PRF* and set the OutputFormat property to 'Pulses', the number of samples can vary.

Default: 10e3

PRFSelectionInputPort

Enable PRF selection input

Enable the PRF selection input, specified as true or false. When you set this property to false, the step method uses the values set in the PRF property. When you set this property to true, you pass an index argument into the step method to select a value from the PRF vector.

Default: false

FrequencyOffsetSource

Source of frequency offset

Source of frequency offset for the waveform, specified as 'Property' or 'Input port'.

- When you set this property to 'Property', the offset is determined by the value of the FrequencyOffset property.
- When you set this property to 'Input port', the FrequencyOffset is determined by the freqoffset input argument.

Default: 'Property'

FrequencyOffset

Frequency offset

Frequency offset in Hz, specified as a scalar.

Dependencies

This property applies when you set the FrequencyOffsetSource property to 'Input port'.

Default: 0 Hz

OutputFormat

Output signal format

Specify the format of the output signal as 'Pulses' or 'Samples'. When you set the OutputFormat property to 'Pulses', the output of the step method takes the form of multiple pulses specified by the value of the NumPulses property. The number of samples per pulse can vary if you change the pulse repetition frequency during the simulation.

When you set the OutputFormat property to 'Samples', the output of the step method is in the form of multiple samples. In this case, the number of output signal samples is the value of the NumSamples property and is fixed.

Default: 'Pulses'

NumSamples

Number of samples in output

Specify the number of samples in the output of the step method as a positive integer. This property applies only when you set the OutputFormat property to 'Samples'.

Default: 100

NumPulses

Number of pulses in output

Specify the number of pulses in the output of the step method as a positive integer. This property applies only when you set the OutputFormat property to 'Pulses'.

Default: 1

PRFOutputPort

Set this property to `true` to output the PRF for the current pulse using a `step` method argument.

Dependencies

This property can be used only when the `OutputFormat` property is set to `'Pulses'`.

Default: `false`

CoefficientsOutputPort

Enable matched filter coefficients output port

Enable the matched filter coefficients output port, specified as `false` or `true`. When you set this property to `false`, the object does not provide the matched filter coefficients used during the simulation as an output. When you set this property to `true`, the object provides the matched filter coefficients used during the simulation as an output.

Default: `false`

Methods

bandwidth	Bandwidth of phase-coded waveform
getMatchedFilter	Matched filter coefficients for waveform
plot	Plot phase-coded pulse waveform
reset	Reset states of phase-coded waveform object
step	Samples of phase-coded waveform

Common to All System Objects	
`release`	Allow System object property value changes

Examples

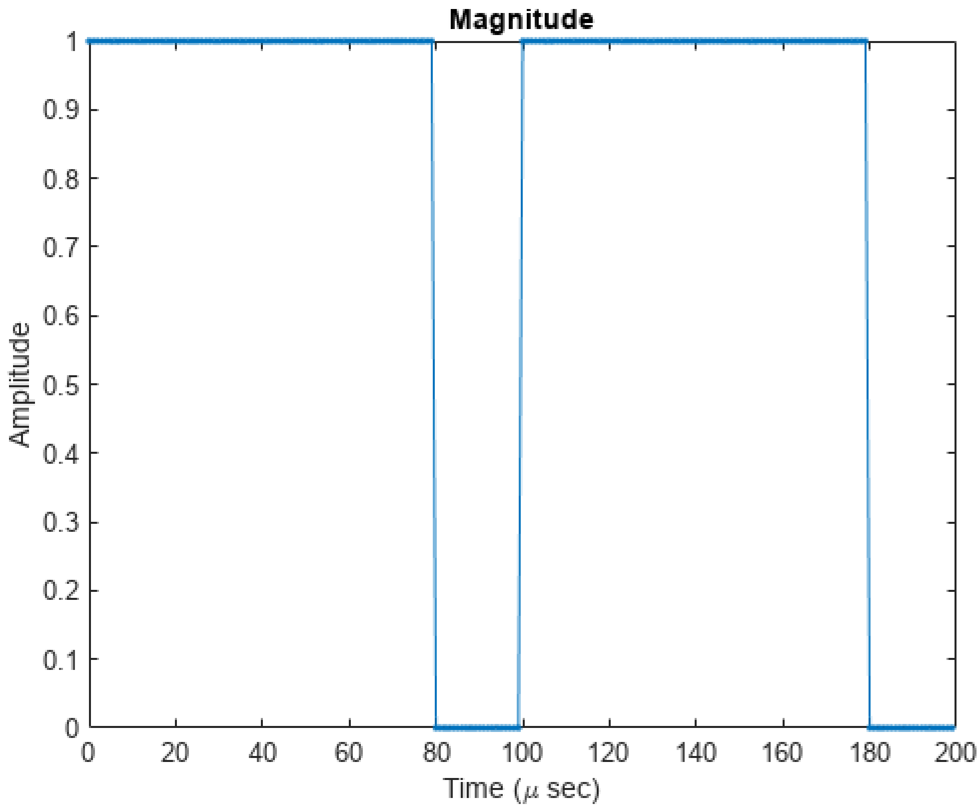
Plot Phase-Coded Waveform and Spectrum

Create and plot a two-pulse phase-coded waveform that uses the Zadoff-Chu code.

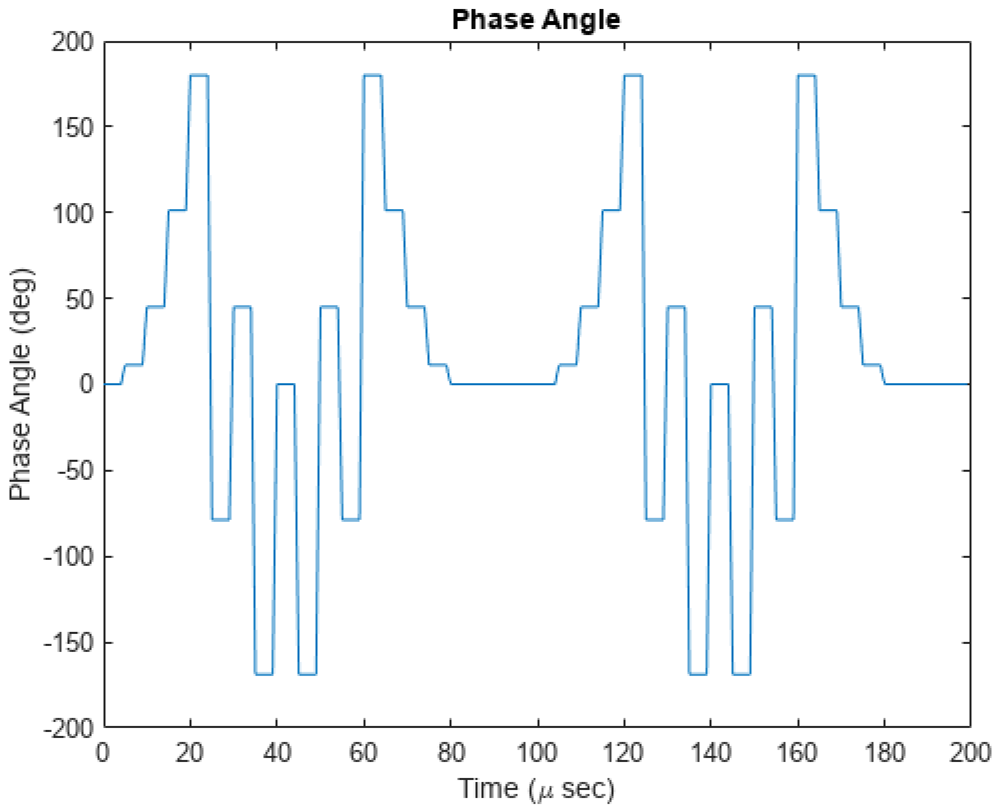
```
sPCW = phased.PhaseCodedWaveform('Code','Zadoff-Chu',...
    'ChipWidth',5e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
fs = sPCW.SampleRate;
```

Generate signal samples and plot the magnitude and phase of the waveforms.

```
wav = step(sPCW);
nsamp = size(wav,1);
t = [0:(nsamp-1)]/fs;
plot(t*1e6,abs(wav),'.-')
title('Magnitude')
xlabel('Time (\mu sec)')
ylabel('Amplitude')
```

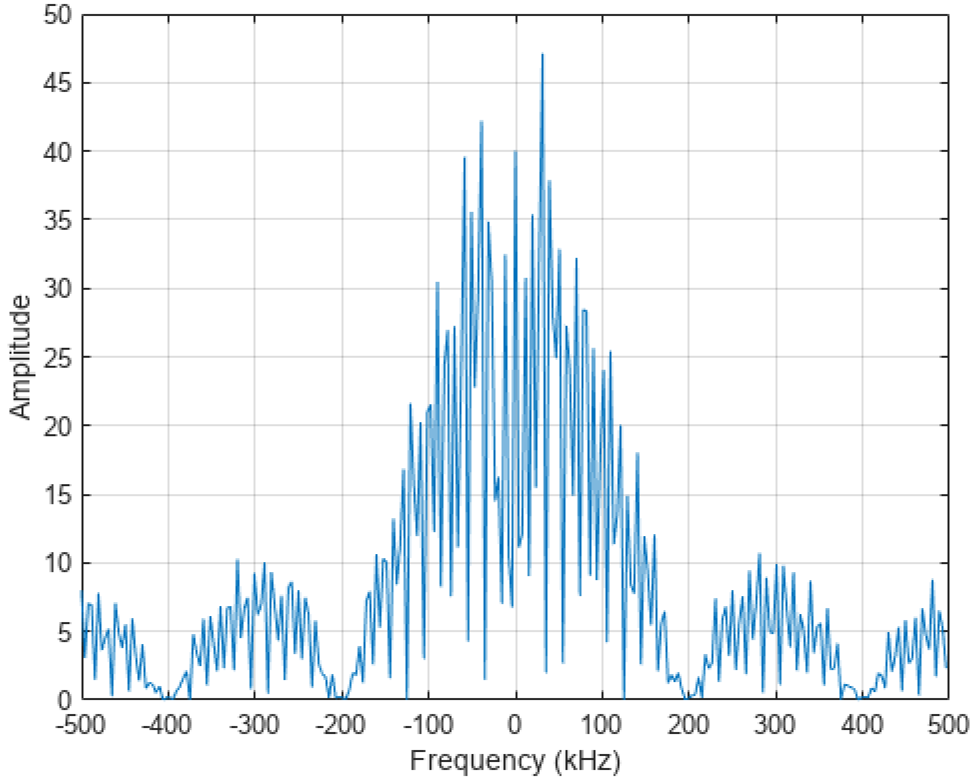



```
plot(t*1e6,180/pi*angle(wav))
title('Phase Angle')
xlabel('Time (\mu sec)')
ylabel('Phase Angle (deg)')
```



Plot the spectrum.

```
nsamp = size(wav,1);
nfft = 2^nextpow2(nsamp);
Z = fft(wav,nfft);
fr = [0:(nfft-1)]/nfft*fs;
fr = fr - fs/2;
plot(fr/1000,abs(fftshift(Z)))
xlabel('Frequency (kHz)')
ylabel('Amplitude')
grid
```



Apply Frequency Offset to Phase-Coded Waveform

Apply a frequency offset to a phase-coded waveform that uses the Zadoff-Chu code. Plot the frequency spectrum of the waveform with and without a frequency offset applied.

Create a phase-coded waveform object which is configured to set the frequency offset from an input when the object is executed.

```
fs = 1e6;
sPCW = phased.PhaseCodedWaveform('SampleRate',fs,'Code','Zadoff-Chu', ...
    'ChipWidth',8e-6,'NumChips',4,'OutputFormat','Pulses', ...
    'NumPulses',1,'FrequencyOffsetSource','Input port');
```

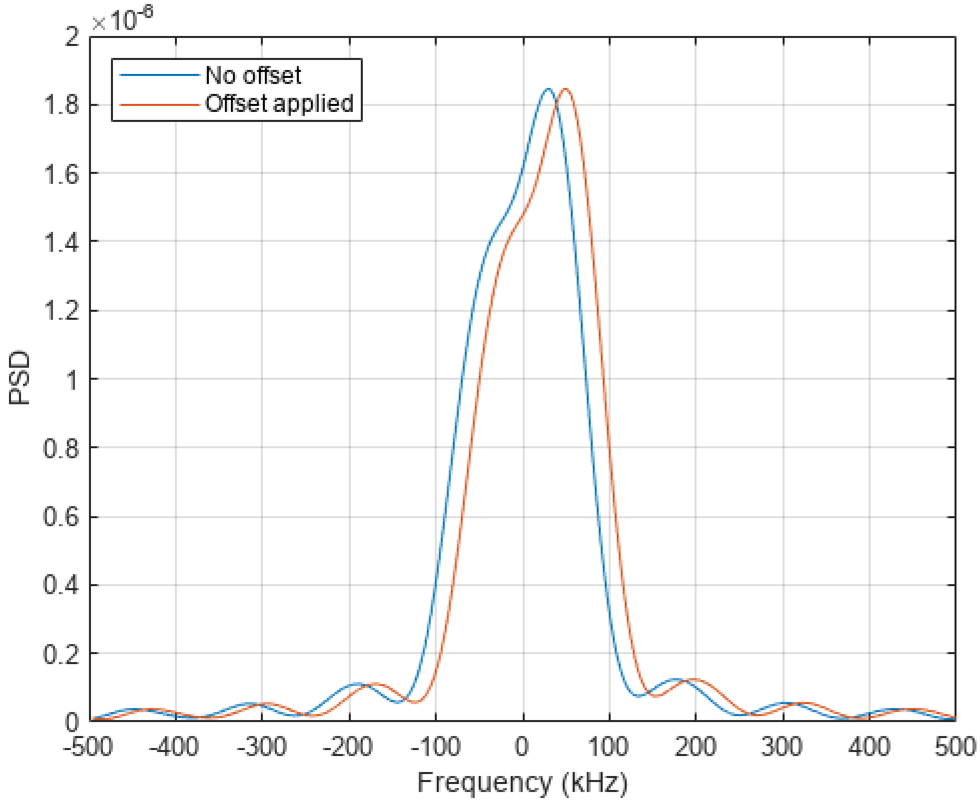
Execute the object two times. First set the frequency offset set to 0 Hz, and then to 2e4 Hz.

```
pcwav = sPCW(0);
pcwav_foffset = sPCW(2e4);
```

Plot the frequency spectrum of the complex signals. The frequency offset signal is shifted to the right.

```
[Pxx,f] = pwelch(pcwav,[],[],[],fs,'centered');
[Pxx_offset,foffset] = pwelch(pcwav_foffset,[],[],[],fs,'centered');
plot(f/1000,Pxx,foffset/1000,Pxx_offset)
ylabel('PSD');
xlabel('Frequency (kHz)');
```

```
legend({'No offset','Offset applied'},'Location','northwest');
grid on;
```



Algorithms

A 2-chip Barker code can use [1 –1] or [1 1] as the sequence of amplitudes. This software implements [1 –1].

A 4-chip Barker code can use [1 1 –1 1] or [1 1 1 –1] as the sequence of amplitudes. This software implements [1 1 –1 1].

A Zadoff-Chu code can use a clockwise or counterclockwise sequence of phases. This software implements the latter, such as $\pi \cdot f(k) \cdot \mathrm{SequenceIndex/NumChips}$ instead of $-\pi \cdot f(k) \cdot \mathrm{SequenceIndex/NumChips}$. In these expressions, k is the index of the chip and $f(k)$ is a function of k.

For further details, see [1].

Version History

Introduced in R2011b

References

[1] Levanon, N. and E. Mozeson. *Radar Signals*. Hoboken, NJ: John Wiley & Sons, 2004.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `plot` method is not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.LinearFMWaveform` | `phased.SteppedFMWaveform` | `phased.RectangularWaveform`

Topics
Waveform Analysis Using the Ambiguity Function
“Phase-Coded Waveforms”

bandwidth

System object: `phased.PhaseCodedWaveform`
Package: `phased`

Bandwidth of phase-coded waveform

Syntax

```
bw = bandwidth(waveform)
```

Description

`bw = bandwidth(waveform)` returns the bandwidth (in hertz) of the pulses for the phase-coded pulse waveform, `waveform`. The bandwidth value is the reciprocal of the chip width.

Input Arguments

`waveform`

Phase-coded waveform object.

Output Arguments

`bw`

Bandwidth of the pulses, in hertz.

Examples

Phase-Coded Waveform Bandwidth

Determine the bandwidth of a Frank phased-coded waveform.

```
waveform = phased.PhaseCodedWaveform;
bw = bandwidth(waveform)
```

```
bw = 1.0000e+05
```

getMatchedFilter

System object: phased.PhaseCodedWaveform
Package: phased

Matched filter coefficients for waveform

Syntax

```
Coeff = getMatchedFilter(H)
Coeff = getMatchedFilter(H,'FrequnecyOffset',FOFFSET)
```

Description

`Coeff = getMatchedFilter(H)` returns the matched filter coefficients for the phase-coded waveform object, `H`. `Coeff` is a column vector.

`Coeff = getMatchedFilter(H,'FrequnecyOffset',FOFFSET)` adds a frequency offset when matched filter coefficients are generated. `FOFFSET` must be a positive scalar. This option is available when you set the `FrequencyOffsetSource` property to `'Input port'` for the input object, `H`.

Input Arguments

`H`

Phase-coded waveform object.

Output Arguments

`Coeff`

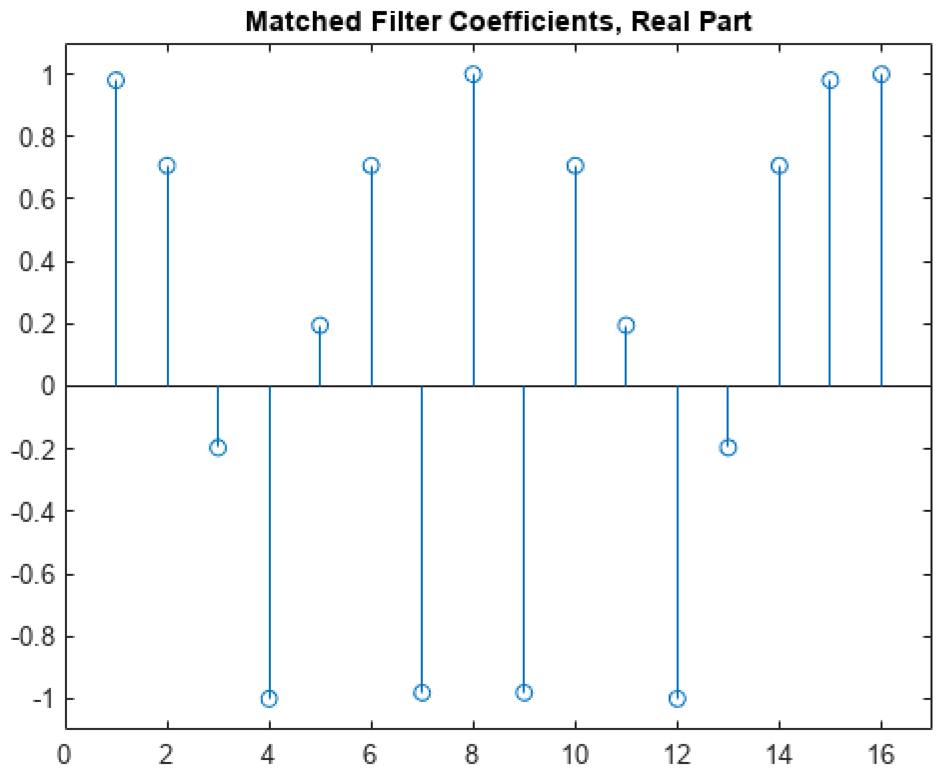
Column vector containing coefficients of the matched filter for `H`.

Examples

Matched-Filter Coefficients for Pulse-Coded Waveform

Obtain the matched filter coefficients for a phase-coded pulse waveform that uses the Zadoff-Chu code.

```
waveform = phased.PhaseCodedWaveform('Code','Zadoff-Chu','ChipWidth',1e-6, ...
    'NumChips',16,'OutputFormat','Pulses','NumPulses',2);
coeff = getMatchedFilter(waveform);
stem(real(coeff))
title('Matched Filter Coefficients, Real Part')
axis([0 17 -1.1 1.1])
```


Matched Filter Coefficients, Real Part
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
2
4
6
8
10
12
14
16

plot

System object: `phased.PhaseCodedWaveform`
Package: `phased`

Plot phase-coded pulse waveform

Syntax

```
plot(Hwav)
plot(Hwav,Name,Value)
plot(Hwav,Name,Value,LineSpec)
h = plot( ___ )
```

Description

`plot(Hwav)` plots the real part of the waveform specified by `Hwav`.

`plot(Hwav,Name,Value)` plots the waveform with additional options specified by one or more `Name,Value` pair arguments.

`plot(Hwav,Name,Value,LineSpec)` specifies the same line color, line style, or marker options as are available in the MATLAB `plot` function.

`h = plot( ___ )` returns the line handle in the figure.

Input Arguments

`Hwav`

Waveform object. This variable must be a scalar that represents a single waveform object.

`LineSpec`

Character vector to specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `PlotType` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Default: `'b'`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PlotType`

Specifies whether the function plots the real part, imaginary part, or both parts of the waveform. Valid values are `'real'`, `'imag'`, and `'complex'`.

Default: `'real'`

`PulseIdx`

Index of the pulse to plot. This value must be a scalar.

Default: `1`

`FrequencyOffset`

Frequency offset

Frequency offset in Hz, specified as a scalar.

Dependencies

This property applies when you set the `FrequencyOffsetSource` property to `'Input port'`.

Default: `0` Hz

Output Arguments

`h`

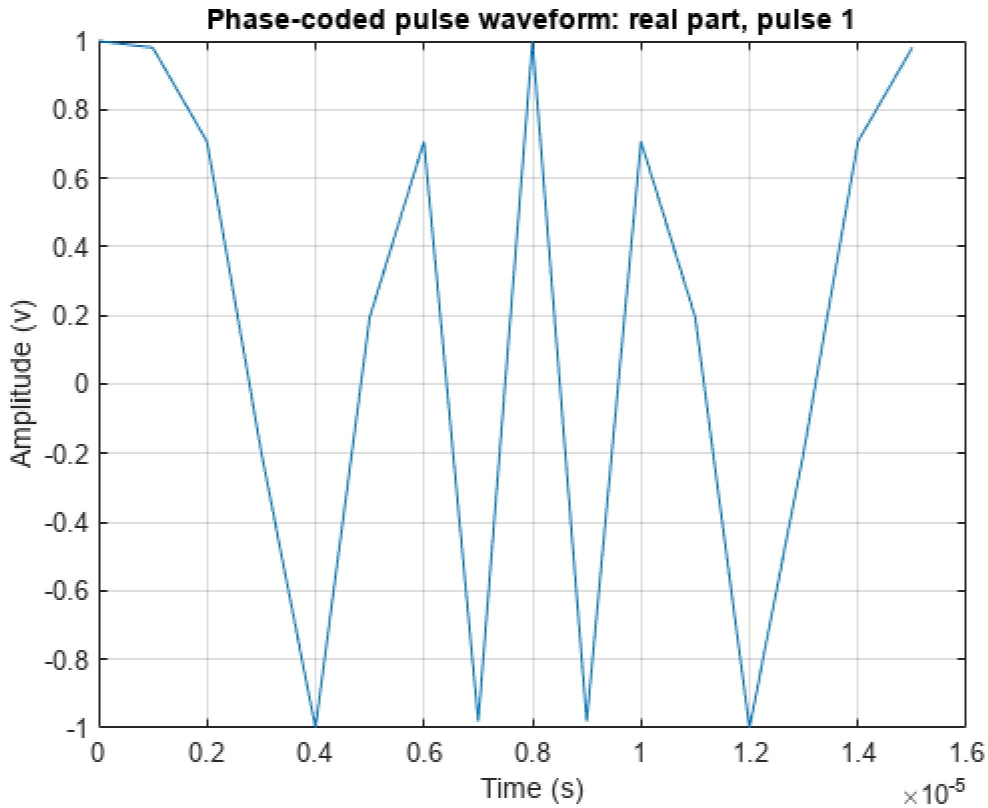
Handle to the line or lines in the figure. For a `PlotType` value of `'complex'`, `h` is a column vector. The first and second elements of this vector are the handles to the lines in the real and imaginary subplots, respectively.

Examples

Plot Pulse-Coded Waveform

Create and plot a phase-coded pulse waveform that uses the Zadoff-Chu code.

```
waveform = phased.PhaseCodedWaveform('Code','Zadoff-Chu','ChipWidth',1e-6, ...
    'NumChips',16,'OutputFormat','Pulses','NumPulses',2);
plot(waveform)
```


Phase-coded pulse waveform: real part, pulse 1
Amplitude (v)
Time (s)
×10^-5
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6

reset

System object: `phased.PhaseCodedWaveform`
Package: `phased`

Reset states of phase-coded waveform object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `PhaseCodedWaveform` object, `H`. Afterward, the next call to `step` restarts the phase sequence from the beginning. Also, if the `PRF` property is a vector, the next call to `step` uses the first PRF value in the vector.

step

System object: `phased.PhaseCodedWaveform`
Package: `phased`

Samples of phase-coded waveform

Syntax

```
Y = step(sPCW)
Y = step(sPCW,prfidx)
Y = step(sRFM,freqoffset)
[Y,PRF] = step( ___ )
[Y,COEFF] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

`Y = step(sPCW)` returns samples of the phase-coded pulse in a column vector, `Y`.

`Y = step(sPCW,prfidx)`, uses the `prfidx` index to select the PRF from the predefined vector of values specified by in the `PRF` property. This syntax applies when you set the `PRFSelectionInputPort` property to `true`.

`Y = step(sRFM,freqoffset)`, uses the `freqoffset` to generate the waveform with an offset as specified at step time. Use this syntax for cases where the transmit pulse frequency needs to be dynamically updated. This syntax applies when you set the `FrequencyOffsetSource` property to `'Input port'`.

`[Y,PRF] = step( ___ )` also returns the current pulse repetition frequency, `PRF`. To enable this syntax, set the `PRFOutputPort` property to `true` and set the `OutputFormat` property to `'Pulses'`.

`[Y,COEFF] = step( ___ )` returns the matched filter coefficients, `COEFF`, for the current pulse. To enable this syntax, set `CoefficientsOutputPort` to `true`. `COEFF` is returned as an N_Z-by-1 vector, where N_Z is the maximum of the nonzero pulse width.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example, `[Y,PRF,COEFF] = step(sRFM,prfidx,freqoffset)`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

sPCW

Phase-coded waveform object.

Output Arguments

Y

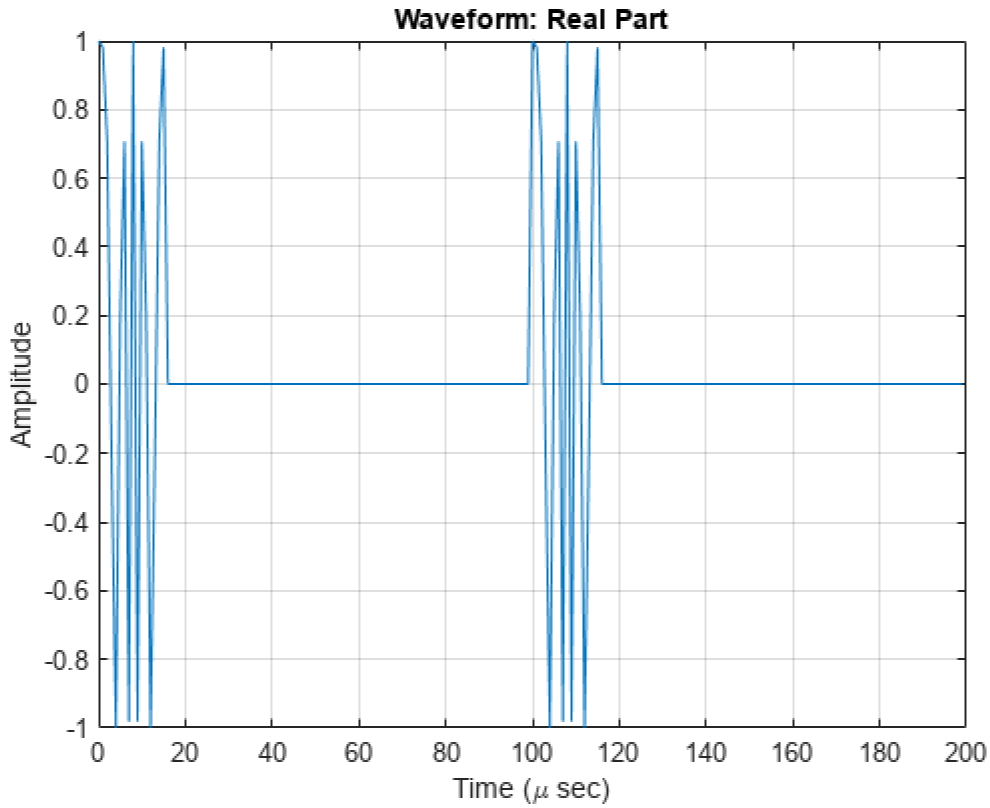
Column vector containing the waveform samples.

Examples

Create Pulse Coded Waveform

Generate samples of two pulses of a phase-coded pulse waveform that uses the Zadoff-Chu code.

```
sPCW = phased.PhaseCodedWaveform('Code','Zadoff-Chu',...
    'ChipWidth',1e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
wav = step(sPCW);
fs = sPCW.SampleRate;
nsamps = size(wav,1);
t = [0:(nsamps-1)]/fs;
plot(t*1e6,real(wav))
title('Waveform: Real Part')
xlabel('Time (\mu sec)')
ylabel('Amplitude')
grid
```



Create Phase-Coded Waveform with Variable PRF

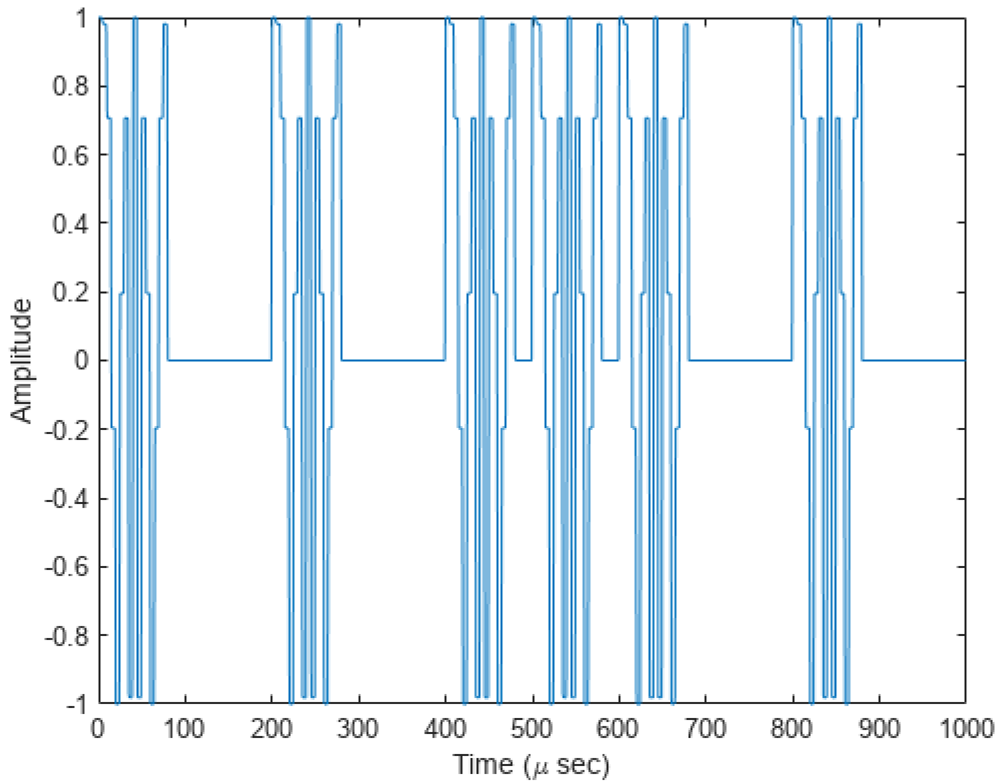
Create and plot two-pulse phase-coded waveforms that uses the Zadoff-Chu code. Set the sample rate to 1 MHz, a chip width of 5 microseconds, 16 chips per pulse. Vary the pulse repetition frequency.

```
fs = 1e6;
PRF = [5000,10000];
waveform = phased.PhaseCodedWaveform('SampleRate',fs,...
    'Code','Zadoff-Chu','PRFSelectionInputPort',true,...
    'ChipWidth',5e-6,'NumChips',16,'PRF',PRF,...
    'OutputFormat','Pulses','NumPulses',2);
```

Obtain and plot the phase-coded waveforms. For the first call to the `step` method, set the PRF to 10kHz using the PRF index. For the next call, set the PRF to 25 kHz. For the final call, set the PRF to 10kHz.

```
wav = [];
wav1 = waveform(1);
wav = [wav; wav1];
wav1 = waveform(2);
wav = [wav; wav1];
wav1 = waveform(1);
wav = [wav; wav1];
nsamps = size(wav,1);
t = [0:(nsamps-1)]/fs;
```

```
plot(t*1e6,real(wav))
xlabel('Time (\mu sec)')
ylabel('Amplitude')
```



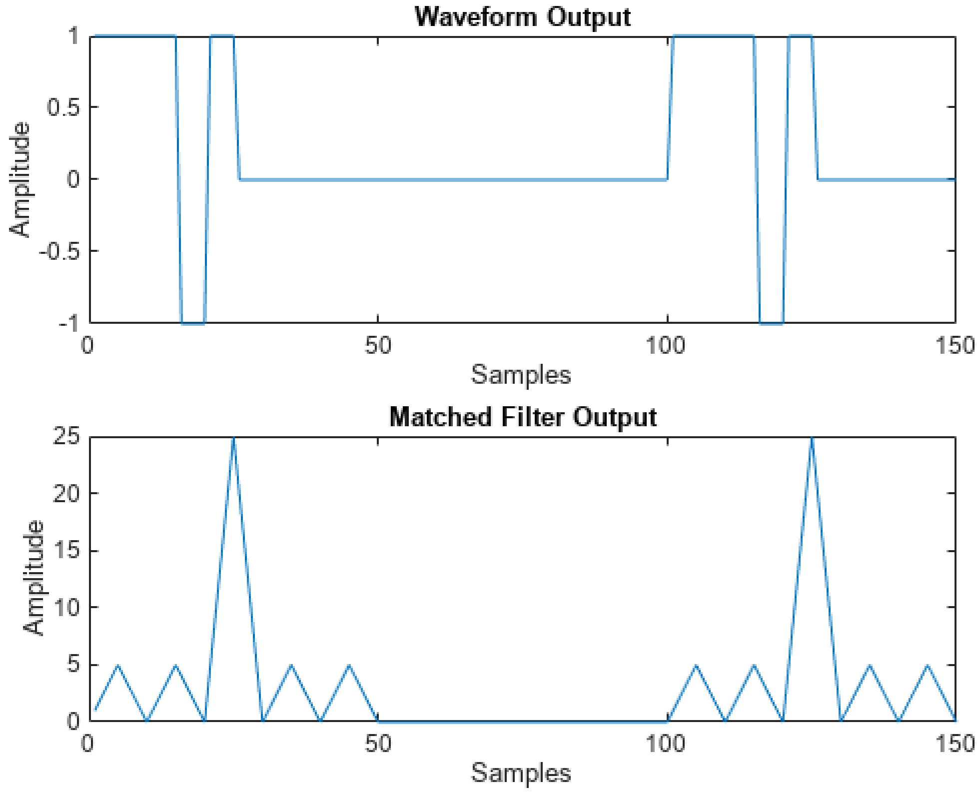
Generate Matched Filter Coefficients of Phase-Coded Waveform

Generate output samples and matched filter coefficients of a Barker coded waveform.

```
waveform = phased.PhaseCodedWaveform('Code','Barker','NumChips',5, ...
    'CoefficientsOutputPort',true,'PRF',[1e4 2e4],'ChipWidth',5e-6, ...
    'OutputFormat','Samples','NumSamples',150);
[wav,coeff] = waveform();
```

Create a matched filter that applies the coefficients as an input argument. Use the coefficients when applying the matched filter to the waveform. Plot the waveform and matched filter outputs.

```
mf = phased.MatchedFilter('CoefficientsSource','Input port');
mfOut = mf(wav,coeff);
subplot(211),plot(real(wav));
xlabel('Samples'),ylabel('Amplitude'),title('Waveform Output');
subplot(212),plot(abs(mfOut));
xlabel('Samples'),ylabel('Amplitude'),title('Matched Filter Output');
```



Waveform Output
Amplitude
1
0.5
0
-0.5
-1
0
50
100
150
Samples
Matched Filter Output
Amplitude
25
20
15
10
5
0
0
50
100
150
Samples

phased.PhaseShiftBeamformer

Package: phased

Narrowband phase shift beamformer

Description

The phased.PhaseShiftBeamformer object implements a narrowband phase-shift beamformer. A phase-shift beamformer approximates a time-delay beamformer for narrowband signals by phase-shifting the arriving signal. A phase shift beamformer belongs to the family of conventional beamformers.

To beamform signals arriving at an array:

1. Create the `phased.PhaseShiftBeamformer` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
beamformer = phased.PhaseShiftBeamformer
beamformer = phased.PhaseShiftBeamformer(Name,Value)
```

Description

`beamformer = phased.PhaseShiftBeamformer` creates a phase-shift beamformer System object, `beamformer`, with default property values.

`beamformer = phased.PhaseShiftBeamformer(Name,Value)` creates a phase-shift beamformer with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `beamformer = phased.PhaseShiftBeamformer('SensorArray',phased.URA,'OperatingFrequency',300e6)` sets the sensor array to a uniform rectangular array (URA) with default URA property values. The beamformer has an operating frequency of 300 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

SensorArray — Sensor array
phased.ULA array with default property values (default) | Phased Array System Toolbox array

Sensor array, specified as an array System object belonging to Phased Array System Toolbox. The sensor array can contain subarrays.

Example: phased.URA

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed').

Example: 3e8

Data Types: single | double

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: 1e9

Data Types: single | double

DirectionSource — Source of beamforming direction
'Property' (default) | 'Input port'

Source of beamforming direction, specified as 'Property' or 'Input port'. Specify whether the beamforming direction comes from the Direction property of this object or from the input argument, ANG. Values of this property are:

'Property'	Specify the beamforming direction using the Direction property.
'Input port'	Specify the beamforming direction using the input argument, ANG.

Data Types: char

Direction — Beamforming directions
[0;0] (default) | real-valued 2-by-1 vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 vector or a real-valued 2-by-*L* matrix. For a matrix, each column specifies a different beamforming direction. Each column has the form [AzimuthAngle;ElevationAngle]. Azimuth angles must lie between –180° and 180° and elevation angles must lie between –90° and 90°. All angles are defined with respect to the local coordinate system of the array. Units are in degrees.

Example: [40;30]

Dependencies

To enable this property, set the DirectionSource property to 'Property'.

Data Types: `single` | `double`

`NumPhaseShifterBits` — Number of phase shifter quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights, specified as a nonnegative integer. A value of zero indicates that no quantization is performed.

Example: 5

Data Types: `single` | `double`

`WeightsNormalization` — Approach for normalizing beamformer weights
`'Distortionless'` (default) | `'Preserve power'`

If you set this property value to `'Distortionless'`, the gain in the beamforming direction is 0 dB. If you set this property value to `'Preserve power'`, the norm of the weights is unity.

Example: `'Preserve power'`

Data Types: `char`

`WeightsOutputPort` — Enable beamforming weights output
`false` (default) | `true`

Enable the output of beamforming weights, specified as `false` or `true`. To obtain the beamforming weights, set this property to `true` and use the corresponding output argument, `W`. If you do not want to obtain the weights, set this property to `false`.

Data Types: `logical`

Usage

Syntax

```
Y = beamformer(X)
Y = beamformer(X,ANG)
[Y,W] = beamformer(___)
```

Description

`Y = beamformer(X)` performs phase-shift beamforming on the input signal, `X`, and returns the beamformed output in `Y`. To use this syntax, set DirectionSource to `'Property'` and set the beamforming direction using the Direction property.

`Y = beamformer(X,ANG)` uses the `ANG` input argument to set the beamforming direction. To use this syntax, set the DirectionSource property to `'Input port'`.

`[Y,W] = beamformer(___)` returns the beamforming weights, `W`. To use this syntax, set the WeightsOutputPort property to `true`.

Input Arguments

`X` — Input signal
complex-valued *M*-by-*N* matrix

Input signal, specified as a complex-valued *M*-by-*N* matrix. If the sensor array contains subarrays, *N* is the number of subarrays; otherwise, *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`
Complex Number Support: Yes

`ANG` — Beamforming directions
`[0;0]` (default) | real-valued 2-by-1 column vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 column vector, or 2-by-*L* matrix. *L* is the number of beamforming directions. Each column has the form `[AzimuthAngle;ElevationAngle]`. Units are in degrees. Each azimuth angle must lie between –180° and 180°, and each elevation angle must lie between –90° and 90°.

Example: `[40;10]`

Dependencies

To enable this argument, set the DirectionSource property to `'Input port'`.

Data Types: `single` | `double`

Output Arguments

`Y` — Beamformed output
complex-valued *M*-by-*L* matrix

Beamformed output, returned as a complex-valued *M*-by-*L* matrix, where *M* is the number of rows of `X` and *L* is the number of beamforming directions.

Data Types: `single` | `double`
Complex Number Support: Yes

`W` — Beamforming weights
complex-valued *N*-by-*L* matrix.

Beamforming weights, returned as a complex-valued *N*-by-*L* matrix. If the sensor array contains subarrays, *N* is the number of subarrays; otherwise, *N* is the number of elements. *L* is the number of beamforming directions.

Dependencies

To enable this output, set the DirectionSource property to `true`.

Data Types: `single` | `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Phase-Shift ULA Beamformer

Apply phase-shift beamforming to a sinewave signal received by a 7-element ULA. The beamforming direction is 45° azimuth and 0° elevation. Assume the array operates at 300 MHz. Specify the beamforming direction using the `Direction` property.

Simulate the signal.

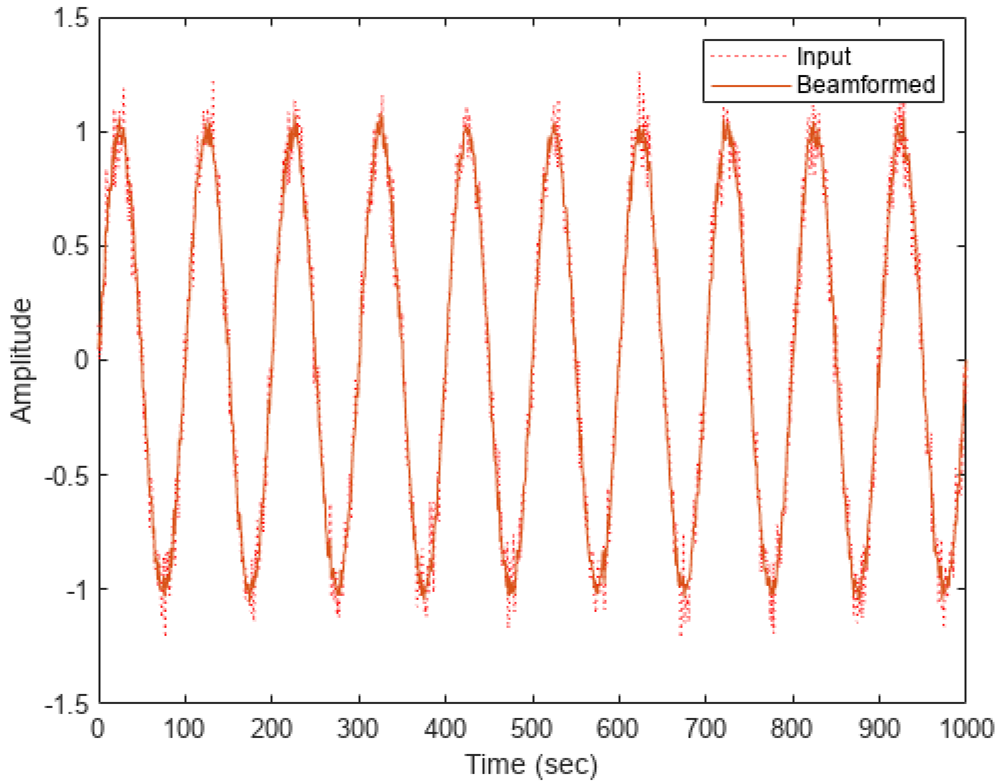
```
t = (0:1000)';
fsignal = 0.01;
x = sin(2*pi*fsignal*t);
c = physconst('Lightspeed');
fc = 300e6;
incidentAngle = [45;0];
array = phased.ULA('NumElements',7);
x = collectPlaneWave(array,x,incidentAngle,fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;
```

Set up a phase-shift beamformer and then beamform the input data.

```
beamformer = phased.PhaseShiftBeamformer('SensorArray',array,...
    'OperatingFrequency',fc,'PropagationSpeed',c,...
    'Direction',incidentAngle,'WeightsOutputPort',true);
[y,w] = beamformer(rx);
```

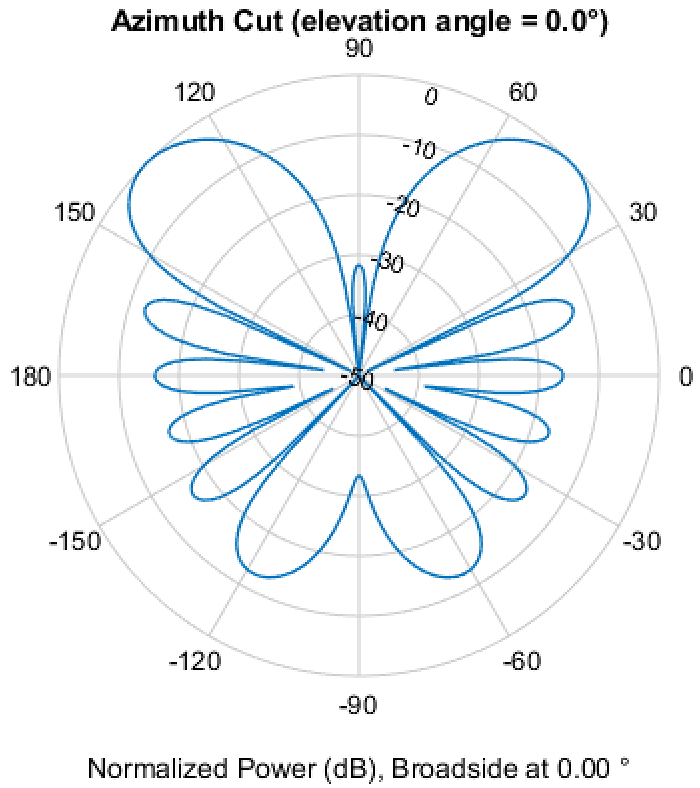
Plot the original signal at the middle element and the beamformed signal.

```
plot(t,real(rx(:,4)),'r:',t,real(y))
xlabel('Time (sec)')
ylabel('Amplitude')
legend('Input','Beamformed')
```



Plot the array response pattern after applying the weights.

```
pattern(array,fc,[-180:180],0,'PropagationSpeed',c,'Type',...
    'powerdb','CoordinateSystem','polar','Weights',w)
```



Phase-Shift Beamformer Using ULA

Apply phase-shift beamforming to the signal received by a 5-element ULA. The beamforming direction is 45° azimuth and 0° elevation. Assume the array operates at 300 MHz. Specify the beamforming direction using an input port.

Simulate a sinewave signal arriving at the array.

```
t = (0:1000)';
fsignal = 0.01;
x = sin(2*pi*fsignal*t);
c = physconst('LightSpeed');
fc = 300e6;
incidentAngle = [45;0];
array = phased.ULA('NumElements',5);
x = collectPlaneWave(array,x,incidentAngle,fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;
```

Construct the phase-shift beamformer and then beamform the input data.

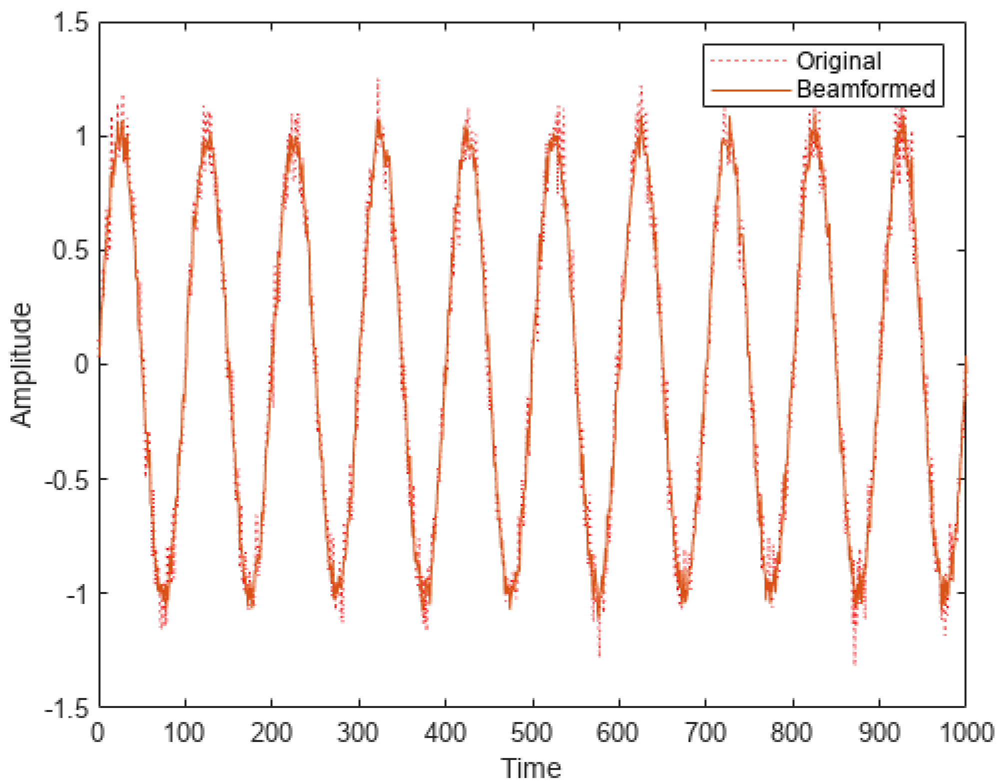
```
beamformer = phased.PhaseShiftBeamformer('SensorArray',array,...
    'OperatingFrequency',fc,'PropagationSpeed',c,...
    'DirectionSource','Input port','WeightsOutputPort',true);
```

Obtain the beamformed signal and the beamformer weights.

```
[y,w] = beamformer(rx,incidentAngle);
```

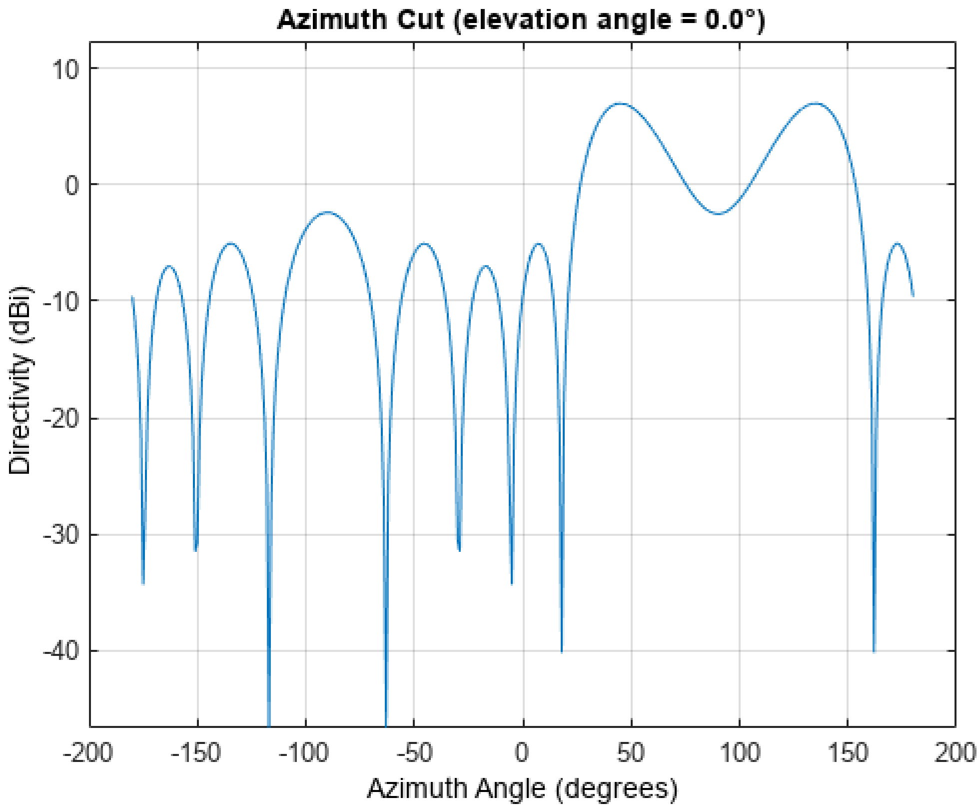
Plot the original signal at the middle element and the beamformed signal.

```
plot(t,real(rx(:,3)),'r:',t,real(y))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Plot the array response pattern after applying the weights.

```
pattern(array,fc,[-180:180],0,'PropagationSpeed',c,'CoordinateSystem','rectangular','Weights',w)
```



Algorithms

Phase Shift Beamforming

The phase shift beamformer uses the conventional delay-and-sum beamforming algorithm. The beamformer assumes the signal is narrowband, so a phase shift can approximate the required delay. The beamformer preserves the incoming signal power.

For more details, see [1].

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York, NY: Wiley-Interscience, 2002.

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.FrostBeamformer` | `phased.MVDRBeamformer` | `phased.LCMVBeamformer` | `phased.SubbandMVDRBeamformer`

step

System object: `phased.PhaseShiftBeamformer`
Package: `phased`

Perform phase shift beamforming

Syntax

```
Y = step(H,X)
Y = step(H,X,ANG)
[Y,W] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` performs phase shift beamforming on the input, `X`, and returns the beamformed output in `Y`.

`Y = step(H,X,ANG)` uses `ANG` as the beamforming direction. This syntax is available when you set the `DirectionSource` property to `'Input port'`.

`[Y,W] = step( ___ )` returns the beamforming weights, `W`. This syntax is available when you set the `WeightsOutputPort` property to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Beamformer object.

X

Input signal, specified as an *M*-by-*N* matrix. If the sensor array contains subarrays, *N* is the number of subarrays; otherwise, *N* is the number of elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

ANG

Beamforming directions, specified as a two-row matrix. Each column has the form [AzimuthAngle; ElevationAngle], in degrees. Each azimuth angle must be between –180 and 180 degrees, and each elevation angle must be between –90 and 90 degrees.

Output Arguments

Y

Beamformed output. Y is an *M*-by-*L* matrix, where *M* is the number of rows of X and *L* is the number of beamforming directions.

W

Beamforming weights. W is an *N*-by-*L* matrix, where *L* is the number of beamforming directions. If the sensor array contains subarrays, *N* is the number of subarrays; otherwise, *N* is the number of elements.

Examples

Phase-Shift Beamformer Using ULA

Apply phase-shift beamforming to the signal received by a 5-element ULA. The beamforming direction is 45° azimuth and 0° elevation. Assume the array operates at 300 MHz. Specify the beamforming direction using an input port.

Simulate a sinewave signal arriving at the array.

```
t = (0:1000)';
fsignal = 0.01;
x = sin(2*pi*fsignal*t);
c = physconst('LightSpeed');
fc = 300e6;
incidentAngle = [45;0];
array = phased.ULA('NumElements',5);
x = collectPlaneWave(array,x,incidentAngle,fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;
```

Construct the phase-shift beamformer and then beamform the input data.

```
beamformer = phased.PhaseShiftBeamformer('SensorArray',array,...
    'OperatingFrequency',fc,'PropagationSpeed',c,...
    'DirectionSource','Input port','WeightsOutputPort',true);
```

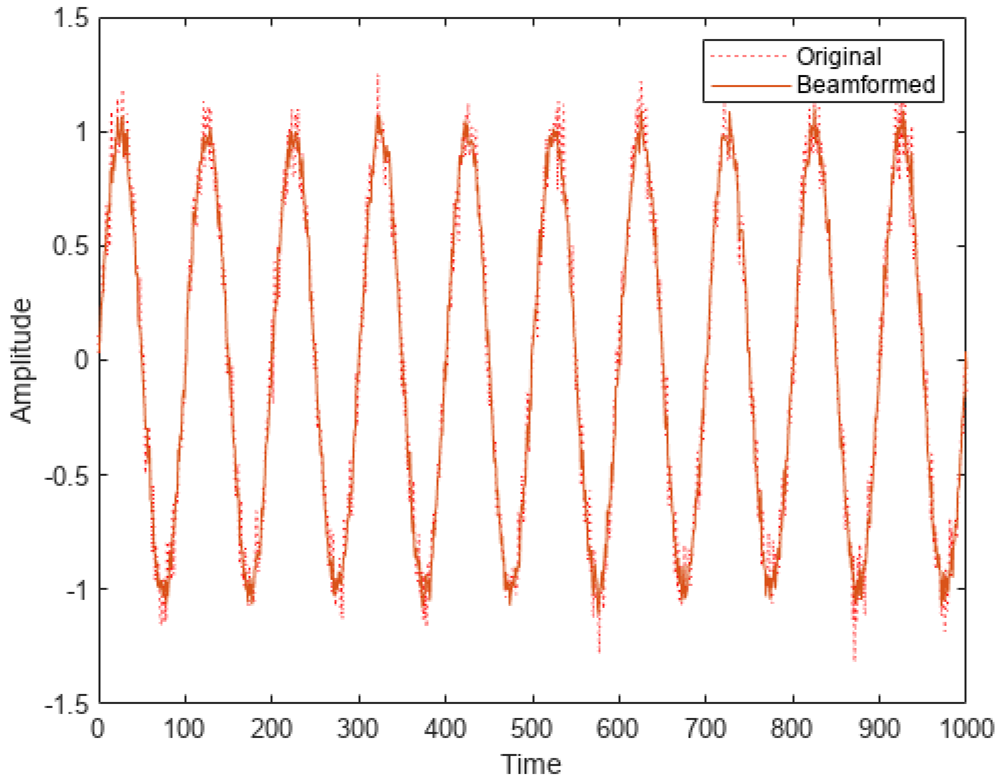
Obtain the beamformed signal and the beamformer weights.

```
[y,w] = beamformer(rx,incidentAngle);
```

Plot the original signal at the middle element and the beamformed signal.

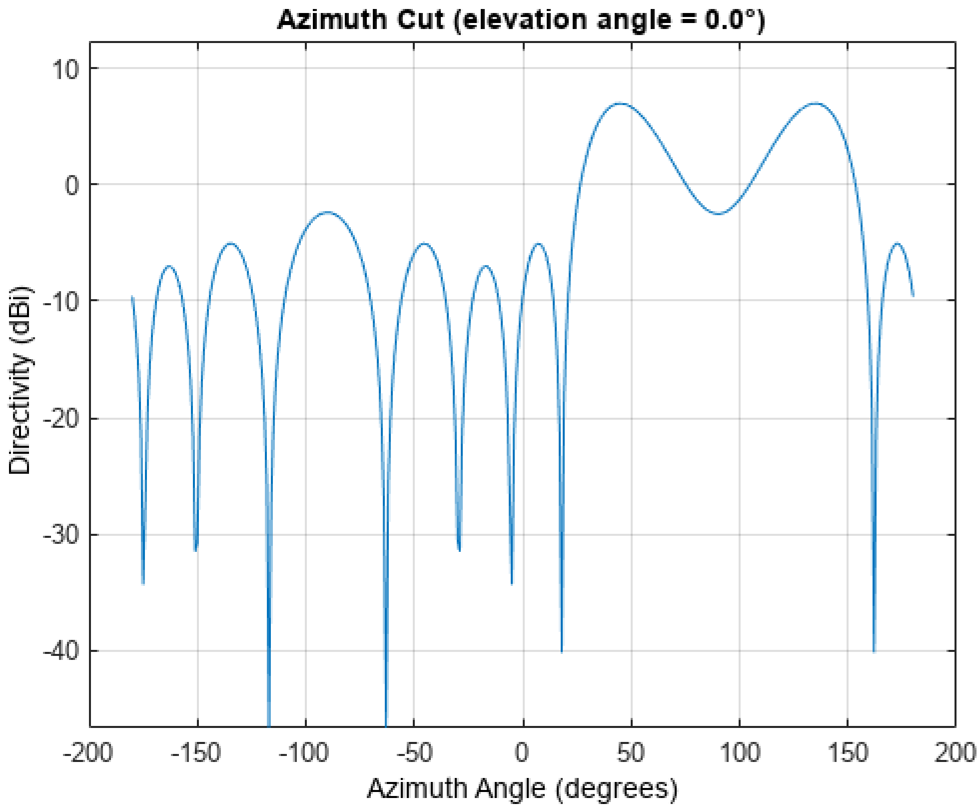
```
plot(t,real(rx(:,3)),'r:',t,real(y))
xlabel('Time')
```

```
ylabel('Amplitude')
legend('Original','Beamformed')
```



Plot the array response pattern after applying the weights.

```
pattern(array,fc,[-180:180],0,'PropagationSpeed',c,'CoordinateSystem','rectangular','Weights',w)
```



Algorithms

The phase shift beamformer uses the conventional delay-and-sum beamforming algorithm. The beamformer assumes the signal is narrowband, so a phase shift can approximate the required delay. The beamformer preserves the incoming signal power.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

phased.Platform

Package: phased

Model platform motion

Description

The phased.Platform System object models the translational motion of one or more platforms in space. A platform can be a target such as a vehicle or airplane, or a sonar or radar transmitter and receiver. The model assumes that the platform undergoes translational motion at constant velocity or constant acceleration during each simulation step. Positions and velocities are always defined in the global coordinate system.

To model a moving platform:

1. Define and set up your platform using the “Construction” on page 1-1240 procedure.
2. Repeatedly call the step method to move the platform along a path determined by the phased.Platform properties.

   The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

sPlat = phased.Platform creates a platform System object, sPlat. The object models a stationary platform with position at the origin and velocity set to zero.

sPlat = phased.Platform(Name,Value) creates an object, sPlat, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

sPlat = phased.Platform(pos,vel,Name,Value) creates a platform object, sPlat, with InitialPosition set to pos and Velocity set to vel. Other specified property Names are set to specified Values. The pos and vel arguments are value-only. Value-only arguments do not require a specified Name but are interpreted according to their argument positions. To specify any value-only argument, specify all preceding value-only arguments.

The motion model is either a constant velocity, a constant acceleration, or a custom trajectory. You can choose one of two motion models using the MotionModel property.

MotionModel Value	Usage
'Velocity'	If you set the VelocitySource property to 'Property', the platform moves with constant velocity determined by the Velocity property. You can specify the InitialPosition property or leave it to its default value. You can change the tunable Velocity property at any simulation step. When you set the VelocitySource property to 'Input port', you can input instantaneous velocity as an argument to the step method. Specify the initial position using the InitialPosition property or leave it as a default value.
'Acceleration'	When you set the AccelerationSource property to 'Property', the platform moves with constant acceleration determined by the Acceleration property. You can specify the InitialPosition and InitialVelocity properties or leave them to their defaults. You can change the tunable Acceleration property at any simulation step. When you set the AccelerationSource property to 'Input port', you can input instantaneous acceleration as an argument to the step method. Specify the InitialPosition and InitialVelocity properties or leave them as their defaults.
'Custom'	Specify the platform motion using a series of waypoints in the CustomTrajectory property.

Properties

MotionModel

Object motion model

Object motion model, specified as 'Velocity', 'Acceleration', or 'Custom'. When you set this property to 'Velocity', the platform follows a constant velocity trajectory during each simulation step. When you set this property to 'Acceleration', the platform follows a constant acceleration trajectory during each simulation step. When you set the property to 'Custom', the platform motion follows a sequence of waypoints specified by the CustomTrajectory property. The object performs a piecewise cubic interpolation on the waypoints to derive the position and velocity at each time step.

Default: 'Velocity'

InitialPosition

Initial position of platform

Initial position of platform, specified as a real-valued 3-by-1 column vector in the form of `[x;y;z]` or a real-valued 3-by-*N* matrix where *N* is the number of platforms. Each column takes the form `[x;y;z]`. Position units are meters.

Default: `[0;0;0]`

`InitialVelocity`

Initial velocity of platform

Initial velocity of platform, specified as a real-valued 3-by-1 column vector in the form of `[vx;vy;vz]` or a real-valued 3-by-*N* matrix where *N* is the number of platforms. Each column taking the form `[vx;vy;vz]`. Velocity units are meters per second.

This property only applies when you set the `MotionModel` property to `'Velocity'` and the `VelocitySource` to `'Input port'`, or when you set the `MotionModel` property to `'Acceleration'`.

Default: `[0;0;0]`

`VelocitySource`

Source of velocity data

Source of velocity data, specified as one of `'Property'` or `'Input port'`. When you set the value of this property to `'Property'`, use `Velocity` property to set the velocity. When you set this property to `'Input port'`, use an input argument of the `step` method to set the velocity.

This property applies when you set the `MotionModel` property to `'Velocity'`.

Default: `'Property'`

`Velocity`

Current velocity of platform

Specify the current velocity of the platform as a 3-by-1 real-valued column vector in the form of `[vx;vy;vz]` or a 3-by-*N* real-valued matrix for multiple platforms. Each column taking the form `[vx;vy;vz]`. Velocity units are meters/sec. The dimension *N* is the number of platforms.

This property applies when you set the `MotionModel` property to `'Velocity'` and the `VelocitySource` to `'Property'`. This property is tunable.

Default: `[0;0;0]`

`AccelerationSource`

Source of acceleration data

Source of acceleration data, specified as one of `'Property'` or `'Input port'`. When you set the value of this property to `'Property'`, specify the acceleration using the `Acceleration` property. When you set this property to `'Input port'`, use an input argument of the `step` method to set the acceleration.

This property applies when you set the `MotionModel` property to `'Acceleration'`.

Default: `'Property'`

`Acceleration`

Acceleration of platform

Specify the current acceleration of the platform as a real-valued 3-by-1 column vector in the form `[ax;ay;az]` or a real-valued 3-by-*N* matrix with each column taking the form `[ax;ay;az]`. The dimension *N* is the number of platforms. Acceleration units are meters/sec/sec.

This property applies when you set the `MotionModel` property to `'Acceleration'` and `AccelerationSource` to `'Property'`. This property is tunable.

Default: `[0;0;0]`

`CustomTrajectory`

Custom trajectory waypoints.

Custom trajectory waypoints, specified as a real-valued *M*-by-*L* matrix, or *M*-by-*L*-by-*N* array. *M* is the number of waypoints. *L* is either 4 or 7.

- When *L* is 4, the first column indicates the times at which the platform position is measured. Columns 2 - 4 are position measurements in x, y, and z coordinates. The velocity is derived from the position measurements.
- When *L* is 7, columns 5 - 7 in the matrix are velocity measurements in x, y, and z coordinates.

When you set the `CustomTrajectory` property to a three-dimensional array, the number of pages, *N*, represent the number of platforms. Time units are in seconds, position units are in meters, and velocity units are in meters per second.

To enable this property, set the `MotionModel` property to `'Custom'`.

`ScanMode`

Mechanical scanning mode

Mechanical scan mode for platform, specified as `'None'`, `'Circular''`, or `'Sector'`, where `'None'` is the default. When you set the `ScanMode` property to `'Circular'`, the platform scan clockwise 360 degrees continuously in the azimuthal direction of the platform orientation axes. When you set the `ScanMode` property to `'Sector'`, the platform scans clockwise in the azimuthal direction in the platform orientation axes within a range specified by the `AzimuthSpan` property. When the platform scan reaches the span limits, the scan reverses direction and scans back to the other scan limit. Scanning happens within the orientation axes of the platform.

`InitialScanAngle`

Initial scan angle of platform

Initial scan angle of platform, specified as a 1-by-*N* vector where *N* is the number of platforms. The scanning occurs in the local coordinate system of the platform. The `InitialOrientationAxes` specifies the original local coordinate system. At the start of the simulation, the orientation axes specified by the `InitialOrientationAxes` are rotated by the angle specified in the `InitialScanAngle` property. The default value is zero. Units are in degrees. This property applies when you set the `ScanMode` property to `'Circular'` or `'Sector'`.

Example: [30 40]

AzimuthSpan

Azimuth span

The azimuth angle span, specified as an *N*-by-2 matrix where *N* is the number of platforms. Each row of the matrix specifies the scan range of the corresponding platform in the form `[ScanAngleLowerBound ScanAngleHigherBound]`. The default value is `[-60 60]`. Units are in degrees. To enable this property, set the `ScanMode` to `'Sector'`.

AzimuthScanRate

Azimuth scan rate

Azimuth scan rate, specified as a 1-by-*N* vector where *N* is the number of platforms. Each entry in the vector is the azimuth scan rate for the corresponding platform. The default value is 10 degrees/second. Units are in degrees/second. To enable this property, set the `ScanMode` property to `'Circular'` or `'Sector'`.

InitialOrientationAxes

Initial orientation axes of platform

Initial orientation axes of platform, specified as a 3-by-3 real-valued orthonormal matrix for a single platform or as a 3-by-3-by-*N* real-valued matrix for multiple platforms. The dimension *N* is the number of platforms. When the orientation matrix is 3-by-3, the three columns represent the axes of the local coordinate system *(xyz)*. When the orientation matrix is 3-by-3-by-*N*, for each *page* index, the resulting 3-by-3 matrix represents the axes of a local coordinate system.

Default: `[1 0 0;0 1 0;0 0 1]`

OrientationAxesOutputPort

Output orientation axes

To obtain the instantaneous orientation axes of the platform, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the orientation axes of the platform, set this property to `false`.

Default: `false`

Methods

reset	Reset platform to initial position
step	Output current position, velocity, and orientation axes of platform

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Simulate Motion of a Platform

Create a platform at the origin having a velocity of (100,100,0) meters per second. Simulate the motion of the platform for two time steps, assuming the time elapsed for each step is one second. The position of the platform is updated after each step.

```
sPlat = phased.Platform([0; 0; 0],[100; 100; 0]);
T = 1;
```

At the first call to step, the position is at its initial value.

```
[pos,v] = step(sPlat,T);
pos
```

```
pos = 3×1

     0
     0
     0
```

At the second call to step, the position changes.

```
[pos,v] = step(sPlat,T);
pos
```

```
pos = 3×1

   100
   100
     0
```

Model Motion of Circling Airplane

Start with an airplane moving along a circular track with a radius of 10 km at a horizontal speed of 100 m/s and descending at a rate of 1 m/sec. To create circular motion, specify a radially-inward acceleration and constrain the acceleration to lie in the horizontal plane. The acceleration of a body moving in a circle is $\frac{v^2}{r}$. The rate of descent is constant. Set the initial orientation axes matrix of the platform to the identity matrix.

Set up the initial conditions

```
alt = 10000;
radcirc = 10000; % 10 km
phi = 60;
initPos = [cosd(phi)*radcirc;sind(phi)*radcirc;alt];
vs = 100.0;
vx = vs*sind(phi);
vy = -vs*cosd(phi);
vz = -1;
initVel = [vx,vy,vz]';
airplane = phased.Platform('MotionModel','Acceleration', ...
    'AccelerationSource','Input port','InitialPosition',initPos, ...
```

```
    'InitialVelocity',initVel,'OrientationAxesOutputPort',true, ...
    'InitialOrientationAxes',eye(3));
accelmag = vs^2/radcirc;
initPos1 = [cosd(phi)*radcirc;sind(phi)*radcirc;0];
unitvec = initPos1/radcirc;
accel = -accelmag*unitvec;
```

Compute the trajectory

Compute the trajectory for 20000 integration steps at T = 0.1 s intervals

```
N = 20000;
tstep = .10;
posmat = zeros(3,N);
for n = 1:N
    [pos,vel,oax] = airplane(tstep,accel);
    velcirc2 = vel(1)^2 + vel(2)^2;
    vmag = sqrt(velcirc2);
    pos1 = [pos(1),pos(2),0]';
    radcirc = sqrt(pos1'*pos1);
    unitvec = pos1/radcirc;
    accelmag = velcirc2/radcirc;
    accel = -accelmag*unitvec;
    posmat(:,n) = pos;
end
```

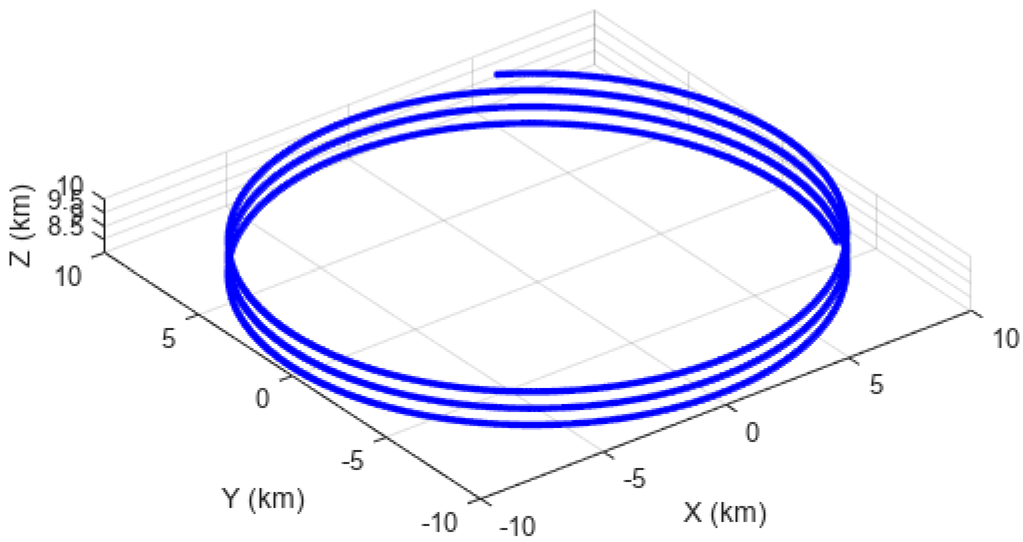
Display the final orientation of the local coordinate system.

```
disp(oax)
```

```
    0.1271    0.9919    0.0001
   -0.9919    0.1271    0.0003
    0.0003   -0.0001    1.0000
```

Plot the trajectory

```
plot3(posmat(1,:)/1000,posmat(2,:)/1000,posmat(3,:)/1000,'b.')
xlabel('X (km)')
ylabel('Y (km)')
zlabel('Z (km)')
axis equal
grid
```



Define Platform Motion Using Waypoints

This example shows

Create waypoints from parabolic motion.

```
x0 = 100;
y0 = -150;
z0 = 0;
vx = 5;
vy = 10;
vz = 0;
ax = 1;
ay = -1;

t = [0:2:20];
x = x0 + vx*t + ax/2*t.^2;
y = y0 + vy*t + ay/2*t.^2;
z = z0*ones(size(t));
wpts = [t.' x.' y.' z.'];
```

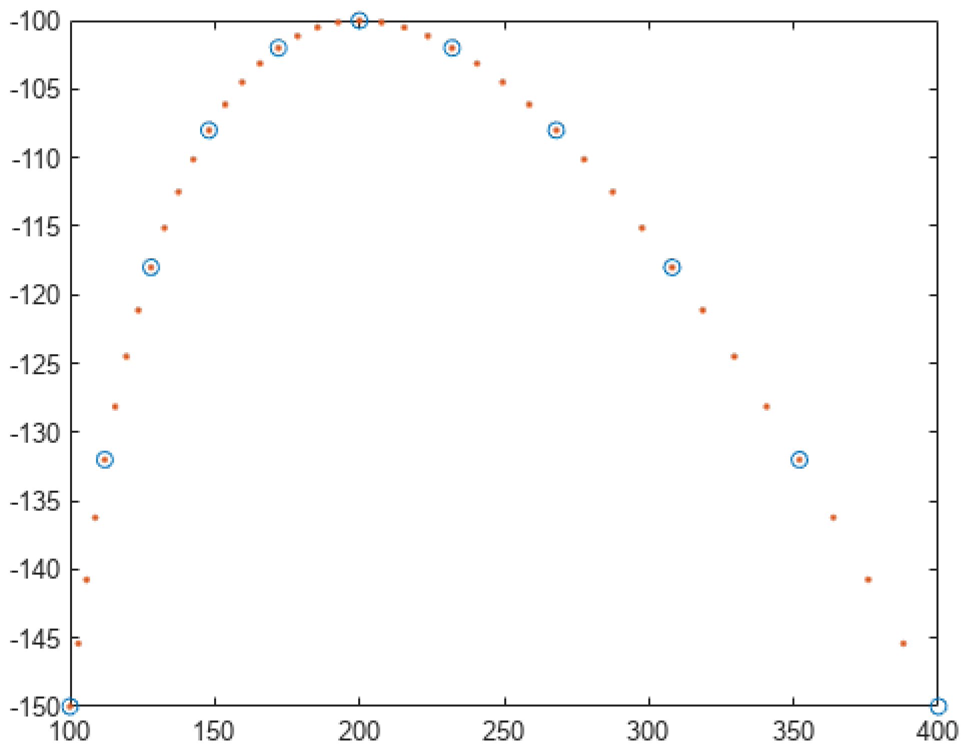
Create a platform object with motion determined using waypoints.

```
pltfm = phased.Platform('MotionModel','Custom','CustomTrajectory',wpts);
tstep = .5;
```

```
nsteps = 40;
X = [];
```

Advance the platform in time steps of one half second;.

```
for k = 1:nsteps
    [pos,vel] = pltfm(tstep);
    X = [X;pos'];
end
plot(x,y,'o'); hold on
plot(X(:,1),X(:,2),'.')
hold off;
```



More About

Platform Orientation

A platform has an associated local coordinate system defined by three orthonormal axis vectors. The direction and magnitude of the velocity vector can change with each call to the `step` method. When the platform undergoes curvilinear motion, the orientation of the local coordinate system axes rotates with the motion of the platform. The change of direction of the velocity vector defines a rotation matrix. The same rotation matrix is then used to rotate the local coordinate system as well. When the velocity vector maintains a constant direction, the rotation matrix is the identity matrix. The initial orientation of the local coordinate system is specified using the `InitialOrientationAxes` property. When you specify multiple platforms, each platform rotates independently.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`global2localcoord` | `local2globalcoord` | `phased.Collector` | `phased.Radiator` | `rangeangle`

Topics

“Motion Modeling in Phased Array Systems”

reset

System object: `phased.Platform`
Package: `phased`

Reset platform to initial position

Syntax

```
reset(H)
```

Description

`reset(H)` resets the initial position of the `Platform` object, `H`.

step

System object: `phased.Platform`
Package: `phased`

Output current position, velocity, and orientation axes of platform

Syntax

```
[Pos,Vel] = step(sPlat,T)
[Pos,Vel] = step(sPlat,T,V)
[Pos,Vel] = step(sPlat,T,A)
[Pos,Vel,Laxes] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[Pos,Vel] = step(sPlat,T)` returns the current position, `Pos`, and velocity, `Vel`, of the platform. The method then updates the position and velocity. When the `MotionModel` property is set to `'Velocity'` and the `VelocitySource` property is set to `'Property'`, the position is updated using the equation *Pos = Pos + Vel*T* where *T* specifies the elapsed time (in seconds) for the current step. When the `MotionModel` property is set to `'Acceleration'` and the `AccelerationSource` property is set to `'Property'`, the position and velocity are updated using the equations *Pos = Pos + Vel*T + 1/2Acl*T^2* and *Vel = Vel + Acl*T* where *T* specifies the elapsed time (in seconds) for the current step.

`[Pos,Vel] = step(sPlat,T,V)` returns the current position, `Pos`, and the current velocity, `Vel`, of the platform. The method then updates the position and velocity using the equation *Pos = Pos + Vel*T* where *T* specifies the elapsed time (in seconds) for the current step. This syntax applies when you set the `MotionModel` property to `'Velocity'` and the `VelocitySource` property to `'Input port'`.

`[Pos,Vel] = step(sPlat,T,A)` returns the current position, `Pos`, and the current velocity, `Vel`, of the platform. The method then updates the position and velocity using the equations *Pos = Pos + Vel*T + 1/2Acl*T^2* and *Vel = Vel + Acl*T* where *T* specifies the elapsed time (in seconds) for the current step. This syntax applies when you set the `MotionModel` property to `'Acceleration'` and the `AccelerationSource` property to `'Input port'`.

`[Pos,Vel,Laxes] = step( ___ )` returns the additional output `Laxes` as the platform's orientation axes when you set the `OrientationAxesOutputPort` property to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

sPlat

Platform

Platform, specified as a `phased.Platform` System object.

T

Step time

Step time, specified as a real-valued scalar. Units are seconds

V

Platform velocity

Platform velocity, specified as a real-valued 3-by-*N* matrix where *N* is the number of platforms to model. This argument applies when you set the `MotionModel` property to `'Velocity'` and the `VelocitySource` property to `'Input port'`. Units are meters per second.

A

Platform acceleration

Platform acceleration, specified as a real-valued 3-by-*N* matrix where *N* is the number of platforms to model. This argument applies when you set the `MotionModel` property to `'Acceleration'` and the `AccelerationSource` property to `'Input port'`. Units are meters per second-squared.

Output Arguments

Pos

Current platform position

Current position of platform, specified as a real-valued 3-by-1 column vector in the form of `[x;y;z]` or a real-valued 3-by-*N* matrix where *N* is the number of platforms to model. Each column takes the form `[x;y;z]`. Units are meters.

Vel

Current platform velocity

Current velocity of platform, specify as a real-valued 3-by-1 column vector in the form of `[vx;vy;vz]` or a real-valued 3-by-*N* matrix where *N* is the number of platforms to model. Each column taking the form `[vx;vy;vz]`. Velocity units are meters per second.

Laxes

Current platform orientation axes

Current platform orientation axes, returned as real-valued 3-by-3-by-*N* matrix where *N* is the number of platforms to model. Each 3-by-3 submatrix is an orthonormal matrix. This output is enabled when you set the `OrientationAxesOutputPort` property to `true`. The current platform axes rotate around the normal vector to the path of the platform.

Examples

Simulate Motion of Two Platforms

Create two moving platforms. The first platform, starting at the origin, has a velocity of (100,100,0) meters per second. The second starts at (1000,0,0) meters and has a velocity of (0,200,0) meters per second. Next, specify different local coordinate axes for each platform defined by rotation matrices. Setting the `OrientationAxesOutputPort` property to `true` lets you retrieve the local coordinate axes at each step.

Set up the platform object.

```
pos0 = [[0;0;0],[1000;0;0]];
vel0 = [[100;100;0],[0;200;0]];
R1 = rotx(30);
R2 = roty(45);
laxes(:,:,1) = R1;
laxes(:,:,2) = R2;
sPlat = phased.Platform(pos0,vel0,...
    'OrientationAxesOutputPort',true,...
    'InitialOrientationAxes',laxes);
```

Simulate the motion of the platform for two time steps, assuming the time elapsed for each step is one second. The position of the platform is updated after each step.

```
T = 1;
```

At the first step, the position and velocity equal the initial values.

```
[pos,v,lax] = step(sPlat,T);
pos
```

```
pos = 3×2

           0        1000
           0           0
           0           0
```

```
lax
```

```
lax = 
lax(:,:,1) =

    1.0000         0         0
         0    0.8660   -0.5000
         0    0.5000    0.8660


lax(:,:,2) =

    0.7071         0    0.7071
         0    1.0000         0
```

```
   -0.7071         0    0.7071
```

At the second step, the position is updated.

```
[pos,v,lax] = step(sPlat,T);
pos
```

```
pos = 3×2

         100        1000
         100         200
           0           0
```

```
lax
```

```
lax = 
lax(:,:,1) =

    1.0000         0         0
         0    0.8660   -0.5000
         0    0.5000    0.8660


lax(:,:,2) =

    0.7071         0    0.7071
         0    1.0000         0
   -0.7071         0    0.7071
```

Free Falling Accelerating Platform

Find the trajectory of a platform which starts with some initial upward velocity but accelerates downward with a constant gravitational acceleration of -9.8 m/sec/sec. Update the platform position and velocity every two seconds.

Construct the platform System object™.

```
platform = phased.Platform('MotionModel','Acceleration','InitialPosition',[2000,100,3000]',...
    'InitialVelocity',[300,150,20]','AccelerationSource','Property','Acceleration',[0,0,-9.8]');
T = 2;
N = 100;
```

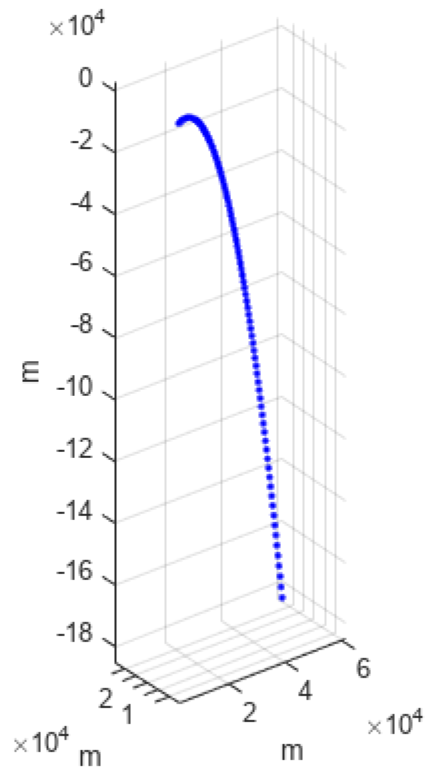
Call the step method for 100 time samples.

```
posmat = zeros(3,N);
for n = 1:N
    [pos,vel] = platform(T);
    posmat(:,n) = pos;
end
```

Plot the trajectory.

```
plot3(posmat(1,:),posmat(2,:),posmat(3,:),'b.')
axis equal
```

```
xlabel('m')
ylabel('m')
zlabel('m')
grid
```



phased.RadarTarget

Package: phased

Radar target

Description

The RadarTarget System object models how a signal is reflected from a radar target. The quantity that determines the response of a target to incoming signals is called the radar target cross-section (RCS). While all electromagnetic radar signals are polarized, you can sometimes ignore polarization and process them as if they were scalar signals. To ignore polarization, specify the EnablePolarization property as false. To utilize polarization, specify the EnablePolarization property as true. For non-polarized processing, the radar cross section is encapsulated in a single scalar quantity called the MeanRCS. For polarized processing, specify the radar cross-section as a 2-by-2 scattering matrix in the ScatteringMatrix property. For both polarization processing types, there are several Swerling models available that can generate random fluctuations in the RCS. Choose these models using the Model property. The SeedSource and Seed properties control the random fluctuations.

The properties that you can use to model the radar cross-section or scattering matrix depend upon the polarization type.

EnablePolarization Value	Use These Properties
false	• MeanRCSSource • MeanRCS
true	• ScatteringMatrixSource • ScatteringMatrix • Mode

To compute the signal reflected from a radar target:

1. Define and set up your radar target. See “Construction” on page 1-1256.
2. Call step to compute the reflected signal according to the properties of phased.RadarTarget. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.RadarTarget creates a radar target System object, H, that computes the reflected signal from a target.

H = phased.RadarTarget(Name,Value) creates a radar target object, H, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

EnablePolarization

Allow polarized signals

Set this property to `true` to allow the target to simulate the reflection of polarized radiation. Set this property to `false` to ignore polarization.

Default: `false`

Mode

Target scattering mode

Target scattering mode specified as one of `'Monostatic'` or `'Bistatic'`. If you set this property to `'Monostatic'`, the reflected signal direction is opposite to its incoming direction. If you set this property to `'Bistatic'`, the reflected direction of the signal differs from its incoming direction. This property applies when you set the `EnablePolarization` property to `true`.

Default: `'Monostatic'`

ScatteringMatrixSource

Sources of mean scattering matrix of target

Source of mean scattering matrix of target specified as one of `'Property'` or `'Input port'`. If you set the `ScatteringMatrixSource` property to `'Property'`, the target's mean scattering matrix is determined by the value of the `ScatteringMatrix` property. If you set this property to `'Input port'`, the mean scattering matrix is determined by an input argument of the `step` method. This property applies only when you set the `EnablePolarization` property to `true`. When the `EnablePolarization` property is set to `false`, use the `MeanRCSSource` property instead, together with the `MeanRCS` property, if needed.

Default: `'Property'`

ScatteringMatrix

Mean radar scattering matrix for polarized signal

Mean radar scattering matrix specified as a complex–valued 2-by-2 matrix. This matrix represents the mean value of the target's radar cross-section. Units are in square meters. The matrix has the form `[s_hh s_hv;s_vh s_vv]`. In this matrix, the component `s_hv` specifies the complex scattering response when the input signal is vertically polarized and the reflected signal is horizontally polarized. The other components are defined similarly. This property applies when you set the `ScatteringMatrixSource` property to `'Property'` and the `EnablePolarization` property to `true`. When the `EnablePolarization` property is set to `false`, use the `MeanRCS` property instead, together with the `MeanRCSSource` property. This property is tunable.

Default: `[1 0;0 1i]`

MeanRCSSource

Source of mean radar cross section

Specify whether the mean RCS value of the target comes from the MeanRCS property of this object or from an input argument in step. Values of this property are:

'Property'	The MeanRCS property of this object specifies the mean RCS value(s).
'Input port'	An input argument in each invocation of step specifies the mean RCS value.

When EnablePolarization property is set to true, use the ScatteringMatrixSource property together with the ScatteringMatrix property.

Default: 'Property'

MeanRCS

Mean radar cross section

Specify the mean value of the target's radar cross section as a nonnegative scalar or as a 1-by-*M* real-valued, nonnegative row vector. Units are in square meters. Using a vector lets you simultaneously process multiple targets. The quantity *M* is the number of targets. This property is used when MeanRCSSource is set to 'Property'. This property is tunable.

When EnablePolarization property is set to true, use the ScatteringMatrix property together with the ScatteringMatrixSource.

Default: 1

Model

Target statistical model

Specify the statistical model of the target as one of 'Nonfluctuating', 'Swerling1', 'Swerling2', 'Swerling3', or 'Swerling4'. If you set this property to a value other than 'Nonfluctuating', you must use the UPDATERCS input argument when invoking step. You can set the mean value of the radar cross-section model by specifying MeanRCS or use its default value.

Default: 'Nonfluctuating'

PropagationSpeed

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar.

Default: Speed of light

OperatingFrequency

Signal carrier frequency

Specify the carrier frequency of the signal you are reflecting from the target, as a scalar in hertz.

Default: 3e8

SeedSource

Source of seed for random number generator

Specify how the object generates random numbers. Values of this property are:

'Auto'	The default MATLAB random number generator produces the random numbers. Use 'Auto' if you are using this object with Parallel Computing Toolbox software.
'Property'	The object uses its own private random number generator to produce random numbers. The Seed property of this object specifies the seed of the random number generator. Use 'Property' if you want repeatable results and are not using this object with Parallel Computing Toolbox software.

The random numbers are used to model random RCS values. This property applies when the Model property is 'Swerling1', 'Swerling2','Swerling3', or 'Swerling4'.

Default: 'Auto'

Seed

Seed for random number generator

Specify the seed for the random number generator as a scalar integer between 0 and 2^{32}–1. This property applies when you set the SeedSource property to 'Property'.

Default: 0

Methods

reset	Reset states of radar target object
step	Reflect incoming signal

Common to All System Objects	
release	Allow System object property value changes

Examples

Compute Reflected Signal from a Non-fluctuating Radar Target

Create a simple signal and compute the value of the reflected signal from a target having a radar cross section of $10m^2$. Set the radar cross section using the MeanRCS property. Set the radar operating frequency to 600 MHz.

```
x = ones(10,1);
target = phased.RadarTarget('Model','Nonfluctuating',...
        'MeanRCS',10,...
        'OperatingFrequency',600e6);
y = target(x);
disp(y(1:3))
```

```
22.4355
22.4355
22.4355
```

This value agrees with the formula $y = \sqrt{G}x$ where

$$G = 4\pi\sigma/\lambda^2$$

Algorithms

For a narrowband nonpolarized signal, the reflected signal, Y, is

$$Y = \sqrt{G} \cdot X,$$

where:

- X is the incoming signal.
- G is the target gain factor, a dimensionless quantity given by

$$G = \frac{4\pi\sigma}{\lambda^2}.$$

  - σ is the mean radar cross-section (RCS) of the target.
  - λ is the wavelength of the incoming signal.

The incident signal on the target is scaled by the square root of the gain factor.

For narrowband polarized waves, the single scalar signal, X, is replaced by a vector signal, (E_H, E_V), with horizontal and vertical components. The scattering matrix, S, replaces the scalar cross-section, σ. Through the scattering matrix, the incident horizontal and vertical polarized signals are converted into the reflected horizontal and vertical polarized signals.

$$\begin{bmatrix} E_H^{(scat)} \\ E_V^{(scat)} \end{bmatrix} = \sqrt{\frac{4\pi}{\lambda^2}} \begin{bmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{bmatrix} \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix} = \sqrt{\frac{4\pi}{\lambda^2}} [S] \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

For further details, see Mott, [1] or Richards, [2] .

Version History

Introduced in R2011a

References

[1] Mott, H., *Antennas for Radar and Communications*, John Wiley & Sons, 1992.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[3] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also
`phased.FreeSpace` | `phased.Platform` | `phased.BackscatterRadarTarget` | `phased.BackscatterSonarTarget` | `phased.WidebandBackscatterRadarTarget` | `backscatterPedestrian`

Topics
“Radar Target”

reset

System object: `phased.RadarTarget`
Package: `phased`

Reset states of radar target object

Syntax

`reset(H)`

Description

`reset(H)` resets the states of the `RadarTarget` object, `H`. This method resets the random number generator state if the `SeedSource` property is applicable and has the value `'Property'`.

step

System object: `phased.RadarTarget`
Package: `phased`

Reflect incoming signal

Syntax

```
Y = step(H,X)
Y = step(H,X,MEANRCS)
Y = step(H,X,UPDATERCS)
Y = step(H,X,MEANRCS,UPDATERCS)

Y = step(H,X,ANGLE_IN,LAXES)
Y = step(H,X,ANGLE_IN,ANGLE_OUT,LAXES)
Y = step(H,X,ANGLE_IN,LAXES,SMAT)
Y = step(H,X,ANGLE_IN,LAXES,UPDATESMAT)
Y = step(H,X,ANGLE_IN,ANGLE_OUT,LAXES,SMAT,UPDATESMAT)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` returns the reflected signal `Y` due to the incident signal `X`. The argument `X` is a complex-valued *N*-by-*1* column vector or *N*-by-*M* matrix. The value *M* is the number of signals. Each signal corresponds to a different target. The value *N* is the number of samples in each signal. Use this syntax when you set the `Model` property of `H` to `'Nonfluctuating'`. In this case, the value of the `MeanRCS` property is used as the Radar cross-section (RCS) value. This syntax applies only when the `EnablePolarization` property is set to `false`. If you specify *M* incident signals, you can specify the radar cross-section as a scalar or as a 1-by-*M* vector. For a scalar, the same value will be applied to all signals.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`Y = step(H,X,MEANRCS)` uses `MEANRCS` as the mean RCS value. This syntax is available when you set the `MeanRCSSource` property to `'Input port'` and set `Model` to `'Nonfluctuating'`. The value of `MEANRCS` must be a nonnegative scalar or 1-by-*M* row vector for multiple targets. This syntax applies only when the `EnablePolarization` property is set to `false`.

`Y = step(H,X,UPDATERCS)` uses `UPDATERCS` as the indicator of whether to update the RCS value. This syntax is available when you set the `Model` property to `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `UPDATERCS` is `true`, a new RCS value is generated. If `UPDATERCS` is `false`, the previous RCS value is used. This syntax applies only when the

EnablePolarization property is set to false. In this case, the value of the MeanRCS property is used as the radar cross-section (RCS) value.

Y = step(H,X,MEANRCS,UPDATERCS) lets you can combine optional input arguments when their enabling properties are set. In this syntax, MeanRCSSource is set to 'Input port' and Model is set to one of the Swerling models. This syntax applies only when the EnablePolarization property is set to false. For this syntax, changes in MEANRCS will be ignored after the first call to the step method.

Y = step(H,X,ANGLE_IN,LAXES) returns the reflected signal Y from an incident signal X. This syntax applies only when the EnablePolarization property is set to true. The input argument, ANGLE_IN, specifies the direction of the incident signal with respect to the target's local coordinate system. The input argument, LAXES, specifies the direction of the local coordinate axes with respect to the global coordinate system. This syntax requires that you set the Model property to 'Nonfluctuating' and the Mode property to 'Monostatic'. In this case, the value of the ScatteringMatrix property is used as the scattering matrix value.

X is a 1-by-*M* row array of MATLAB struct type, each member of the array representing a different signal. The struct contains three fields, X.X, X.Y, and X.Z. Each field corresponds to the *x*, *y*, and *z* components of the polarized input signal. Polarization components are measured with respect to the global coordinate system. Each field is a column vector representing a sequence of values for each incoming signal. The X.X, X.Y, and Y.Z fields must all have the same dimension. The argument, ANGLE_IN, is a 2-by-*M* matrix representing the signals' incoming directions with respect to the target's local coordinate system. Each column of ANGLE_IN specifies the incident direction of the corresponding signal in the form [AzimuthAngle; ElevationAngle]. Angle units are in degrees. The number of columns in ANGLE_IN must equal the number of signals in the X array. The argument, LAXES, is a 3-by-3 matrix. The columns are unit vectors specifying the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively, with respect to the global coordinate system. Each column is written in [x;y;z] form.

Y is a row array of struct type having the same size as X. Each struct contains the three reflected polarized fields, Y.X, Y.Y, and Y.Z. Each field corresponds to the *x*, *y*, and *z* component of the signal. Polarization components are measured with respect to the global coordinate system. Each field is a column vector representing one reflected signal.

The size of the first dimension of the matrix fields within the struct can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

Y = step(H,X,ANGLE_IN,ANGLE_OUT,LAXES), in addition, specifies the reflection angle, ANGLE_OUT, of the reflected signal when you set the Mode property to 'Bistatic'. This syntax applies only when the EnablePolarization property is set to true. ANGLE_OUT is a 2-row matrix representing the reflected direction of each signal. Each column of ANGLE_OUT specifies the reflected direction of the signal in the form [AzimuthAngle; ElevationAngle]. Angle units are in degrees. The number of columns in ANGLE_OUT must equal the number of members in the X array. The number of columns in ANGLE_OUT must equal the number of elements in the X array.

Y = step(H,X,ANGLE_IN,LAXES,SMAT) specifies SMAT as the scattering matrix. This syntax applies only when the EnablePolarization property is set to true. The input argument SMAT is a 2-by-2 matrix. You must set the ScatteringMatrixSource property 'Input port' to use SMAT.

Y = step(H,X,ANGLE_IN,LAXES,UPDATESMAT) specifies UPDATESMAT to indicate whether to update the scattering matrix when you set the Model property to 'Swerling1', 'Swerling2'', 'Swerling3', or 'Swerling4'. This syntax applies only when the EnablePolarization property is set to true. If UPDATESMAT is set to true, a scattering matrix value is generated. If UPDATESMAT is false, the previous scattering matrix value is used.

Y = step(H,X,ANGLE_IN,ANGLE_OUT,LAXES,SMAT,UPDATESMAT). You can combine optional input arguments when their enabling properties are set. Optional inputs must be listed in the same order as the order of their enabling properties.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Examples

Compute Reflected Signals from Two Non-Fluctuating Radar Targets

Create two sinusoidal signals and compute the value of the reflected signals from targets having radar cross sections of $5m^2$ and $10m^2$, respectively. Set the radar cross sections in the step method by choosing Input port for the value of the MeanRCSSource property. Set the radar operating frequency to 600 MHz.

```
sRadarTarget = phased.RadarTarget('Model','Nonfluctuating',...
        'MeanRCSSource','Input port',...
        'OperatingFrequency',600e6);
t = linspace(0,1,1000);
x = [cos(2*pi*250*t)',10*sin(2*pi*250*t)'];
y = step(sRadarTarget,x,[5,10]);
disp(y(1:3,1:2))
```

```
   15.8643         0
   -0.0249  224.3546
  -15.8642   -0.7055
```

Algorithms

For a narrowband nonpolarized signal, the reflected signal, Y, is

$$Y = \sqrt{G} \cdot X,$$

where:

- X is the incoming signal.
- G is the target gain factor, a dimensionless quantity given by

$$G = \frac{4\pi\sigma}{\lambda^2}.$$

  - σ is the mean radar cross-section (RCS) of the target.
  - λ is the wavelength of the incoming signal.

The incident signal on the target is scaled by the square root of the gain factor.

For narrowband polarized waves, the single scalar signal, *X*, is replaced by a vector signal, (E_H, E_V), with horizontal and vertical components. The scattering matrix, *S*, replaces the scalar cross-section, σ. Through the scattering matrix, the incident horizontal and vertical polarized signals are converted into the reflected horizontal and vertical polarized signals.

$$\begin{bmatrix} E_H^{(scat)} \\ E_V^{(scat)} \end{bmatrix} = \sqrt{\frac{4\pi}{\lambda^2}} \begin{bmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{bmatrix} \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix} = \sqrt{\frac{4\pi}{\lambda^2}} [S] \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

For further details, see Mott [1] or Richards[2].

References

[1] Mott, H. *Antennas for Radar and Communications*.John Wiley & Sons, 1992.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[3] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

See Also

Topics
“Swerling 1 Target Models”
“Swerling Target Models”
“Swerling 3 Target Models”
“Swerling 4 Target Models”

phased.Radiator

Package: `phased`

Narrowband signal radiator

Description

The `phased.Radiator` System object implements a narrowband signal radiator. A radiator converts signals into radiated wavefields transmitted from arrays and individual sensor elements such as antennas, microphone elements, and sonar transducers. The radiator output represents the fields at a reference distance of one meter from the phase center of the element or array. You can then propagate the signals to the far field using, for example, the `phased.FreeSpace`, `phased.LOSChannel`, or `twoRayChannel` System objects.

The object radiates fields in one of two ways controlled by the CombineRadiatedSignals property.

- If the CombineRadiatedSignals is set to `true`, the radiated field in a specified directions is the coherent sum of the delayed radiated fields from all elements (or subarrays when subarrays are supported). The object uses the phase-shift approximation of time delays for narrowband signals.
- If the CombineRadiatedSignals is set to `false`, each element can radiate in an independent direction.

You can use this object to

- model electromagnetic radiated signals as polarized or non-polarized fields depending upon whether the element or array supports polarization and the value of the “Polarization” on page 1-0 property. Using polarization, you can transmit a signal as a polarized electromagnetic field, or transmit two independent signals using dual polarizations.
- model acoustic radiated fields by using nonpolarized microphone and sonar transducer array elements and by setting the “Polarization” on page 1-0 to `'None'`. You must also set the PropagationSpeed to a value appropriate for the medium.
- radiate fields from subarrays created by the `phased.ReplicatedSubarray` and `phased.PartitionedArray` objects. You can steer all subarrays in the same direction using the steering angle argument, `STEERANG`, or steer each subarray in a different direction using the Subarray element weights argument, `WS`. The radiator distributes the signal powers equally among the elements of each subarray. You cannot set the CombineRadiatedSignals property to `false` for subarrays.

To radiate signals:

1. Create the `phased.Radiator` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
radiator = phased.Radiator
radiator = phased.Radiator(Name,Value)
```

Description

`radiator = phased.Radiator` creates a narrowband signal radiator object, `radiator`, with default property values.

`radiator = phased.Radiator(Name,Value)` creates a narrowband signal radiator with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `radiator = phased.Radiator('Sensor',phased.URA,'OperatingFrequency',300e6)` sets the sensor array to a uniform rectangular array (URA) with default URA property values. The beamformer has an operating frequency of 300 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Sensor` — Sensor element or sensor array
`phased.ULA` array with default property values (default) | Phased Array System Toolbox sensor or array

Sensor element or sensor array, specified as a System object belonging to Phased Array System Toolbox. A sensor array can contain subarrays.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `double`

`CombineRadiatedSignals` — Combine radiated signals
`true` (default) | `false`

Combine radiated signals, specified as `true` or `false`. This property enables the coherent summation of the radiated signals from all elements of an array to produce plane waves. Set this property to `false` to obtain individual radiated signal for each radiating element.

- If the CombineRadiatedSignals is set to `true`, the radiated field in a specified directions is the coherent sum of the delayed radiated fields from all elements (or subarrays when subarrays are supported). The object uses the phase-shift approximation of time delays for narrowband signals.
- If the CombineRadiatedSignals is set to `false`, each element can radiate in an independent direction. If the Sensor property is an array that contains subarrays, you cannot set the CombineRadiatedSignals property to `'false`.

Data Types: `logical`

`SensorGainMeasure` — Specify sensor gain
`'dB'` (default) | `'dBi'`

Sensor gain measure, specified as `'dB'` or `'dBi'`.

- When you set this property to `'dB'`, the input signal power is scaled by the sensor power pattern (in dB) at the corresponding direction and then combined.
- When you set this property to `'dBi'`, the input signal power is scaled by the directivity pattern (in dBi) at the corresponding direction and then combined. This option is useful when you want to compare results with the values computed by the radar equation that uses dBi to specify the antenna gain. The computation using the `'dBi'` option is expensive as it requires an integration over all directions to compute the total radiated power of the sensor.

Dependencies

To enable this property, set the `CombineRadiatedSignals` property to `true`.

Data Types: `char`

`Polarization` — Polarization configuration
`'None'` (default) | `'Combined'` | `'Dual'`

Polarization configuration, specified as `'None'`, `'Combined'`, or `'Dual'`. When you set this property to `'None'`, the output field is considered a scalar field. When you set this property to `'Combined'`, the radiated fields are polarized and are interpreted as a single signal in the sensor's inherent polarization. When you set this property to `'Dual'`, the *H* and *V* polarization components of the radiated field are independent signals.

Example: `'Dual'`

Data Types: `char`

`WeightsInputPort` — Enable weights input
`false` (default) | `true`

Enable weights input, specified as `false` or `true`. When `true`, use the object input argument `W` to specify weights. Weights are applied to individual array elements (or at the subarray level when subarrays are supported).

Data Types: `logical`

Usage

Syntax

```
Y = radiator(X,ANG)
Y = radiator(X,ANG,LAXES)
Y = radiator(XH,XV,ANG,LAXES)
Y = radiator( ___ ,W)
Y = radiator( ___ ,STEERANG)
Y = radiator( ___ ,WS)
Y = radiator(X,ANG,LAXES,W,STEERANG)
```

Description

`Y = radiator(X,ANG)` radiates the fields, `Y`, derived from signals, `X` in the directions specified by `ANG`.

`Y = radiator(X,ANG,LAXES)` also specifies `LAXES` as the local coordinate system axes directions. To use this syntax, set the "Polarization" on page 1-0 property to `'Combined'`.

`Y = radiator(XH,XV,ANG,LAXES)` specifies a horizontal-polarization port signal, `XH`, and a vertical-polarization port signal, `XV`. To use this syntax, set the "Polarization" on page 1-0 property to `'Dual'`.

`Y = radiator( ___ ,W)` also specifies `W` as element or subarray weights. To use this syntax, set the WeightsInputPort property to `true`.

`Y = radiator( ___ ,STEERANG)` also specifies `STEERANG` as the subarray steering angle. To use this syntax, set the Sensor property to an array that supports subarrays and set the `SubarraySteering` property of that array to either `'Phase'` or `'Time'`.

`Y = radiator( ___ ,WS)` also specifies `WS` as weights applied to each element within each subarray. To use this syntax, set the Sensor property to an array that supports subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

You can combine optional input arguments when their enabling properties are set, for example, `Y = radiator(X,ANG,LAXES,W,STEERANG)` combines several input arguments. Optional inputs must be listed in the same order as the order of the enabling properties.

Input Arguments

X — Signal to radiate

complex-valued M-by-1 vector | complex-valued M-by-N matrix

Signal to radiate, specified as a complex-valued M-by-1 vector or complex-valued M-by-N matrix. M is the length of the signal, and N is the number of array elements (or subarrays when subarrays are supported).

Dimensions of X

Dimension	Signal
M-by-1 vector	The same signal is radiated from all array elements (or all subarrays when subarrays are supported).
`M-by-N matrix`	Each column corresponds to the signal radiated by the corresponding array element (or corresponding subarrays when subarrays are supported).

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this argument, set the Polarization property to `'None'` or `'Combined'`.

Data Types: `double`
Complex Number Support: Yes

`ANG` — Radiating directions of signals

real-valued 2-by-*L* matrix

Radiating directions of signals, specified as a real-valued 2-by-*L* matrix. Each column specifies a radiating direction in the form `[AzimuthAngle;ElevationAngle]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive. When the CombineRadiatedSignals property is `false`, the number of angles must equal the number of array elements, *N*. Units are in degrees.

Example: `[30,20;45,0]`

Data Types: `double`

`LAXES` — Local coordinate system

real-valued 3-by-3 orthogonal matrix

Local coordinate system, specified as a real-valued 3-by-3 orthogonal matrix. The matrix columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes with respect to the global coordinate system.

Example: `rotx(30)`

Dependencies

To enable this argument, set the `Polarization` property to `'Combined'` or `'Dual'`.

Data Types: `double`

`XH` — H-polarization port signal to radiate

complex-valued *M*-by-1 vector | complex-valued *M*-by-*N* matrix

H-polarization port signal to radiate, specified as a complex-valued *M*-by-1 vector or complex-valued *M*-by-*N* matrix. *M* is the length of the signal, and *N* is the number of array elements (or subarrays when subarrays are supported).

Dimensions of XH

Dimension	Signal
M-by-1 vector	The same signal is radiated from all array elements (or all subarrays when subarrays are supported).
`M-by-N matrix`	Each column corresponds to the signal radiated by the corresponding array element (or corresponding subarrays when subarrays are supported).

The dimensions and sizes of XH and XV must be the same.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this argument, set the Polarization property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

XV — V-polarization port signal to radiate
complex-valued *M*-by-1 vector | complex-valued *M*-by-*N* matrix

V-polarization port signal to radiate, specified as a complex-valued *M*-by-1 vector or complex-valued *M*-by-*N* matrix. *M* is the length of the signal, and *N* is the number of array elements (or subarrays when subarrays are supported).

Dimensions of XV

Dimension	Signal
M-by-1 vector	The same signal is radiated from all array elements (or all subarrays when subarrays are supported).
`M-by-N matrix`	Each column corresponds to the signal radiated by the corresponding array element (or corresponding subarrays when subarrays are supported).

The dimensions and sizes of XH and XV must be the same.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this argument, set the Polarization property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

W — Element or subarray weights
N-by-1 column vector

Element or subarray weights, specified as a complex-valued N-by-1 column vector where N is the number of array elements (or subarrays when the array supports subarrays).

Dependencies

To enable this argument, set the WeightsInputPort property to `true`.

Data Types: `double`
Complex Number Support: Yes

WS — Subarray element weights
complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray element weights

Sensor Array	Subarray weights
`phased.ReplicatedSubarray`	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of `WS` specifies the weights for the corresponding subarray.
`phased.PartitionedArray`	Subarrays may not have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `Sensor` property to an array that contains subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

STEERANG — Subarray steering angle
real-valued 2-by-1 vector

Subarray steering angle, specified as a length-2 column vector. The vector has the form `[azimuthAngle;elevationAngle]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive. Units are in degrees.

Example: `[20;15]`

Dependencies

To enable this argument, set the `Sensor` property to an array that supports subarrays and set the `SubarraySteering` property of that array to either `'Phase'` or `'Time'`

Data Types: `double`

Output Arguments

Y — Radiated signals

complex-valued *M*-by-*L* matrix | complex-valued 1-by-*L* cell array of structures

Radiated signals, specified as a complex-valued *M*-by-*L* matrix or a 1-by-*L* cell array, where *L* is the number of radiating angles, `ANG`. *M* is the length of the input signal, `X`.

- If the Polarization property value is set to `'None'`, the output argument `Y` is an *M*-by-*L* matrix.
- If the Polarization property value is set to `'Combined'` or `'Dual'`, `Y` is a 1-by-*L* cell array of structures. Each cell corresponds to a separate radiating signal. Each `struct` contains three column vectors containing the *X*, *Y*, and *Z* components of the polarized fields defined with respect to the global coordinate system.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Radiation from 5-Element ULA

Propagate and combine radiation from five isotropic antenna elements. Set up a uniform line array of five isotropic antennas.

First construct a ULA array.

```
array = phased.ULA('NumElements',5);
```

Construct a radiator object.

```
radiator = phased.Radiator('Sensor',array,...
    'OperatingFrequency',300e6,'CombineRadiatedSignals',true);
```

Create a simple signal to radiate.

```
x = [1;-1;1;-1;1;-1];
```

Specify the azimuth and elevation of the radiating direction.

```
radiatingAngle = [30;10];
```

Radiate the signal.

```
y = radiator(x,radiatingAngle)
```

```
y = 6×1 complex

  -0.9523 - 0.0000i
   0.9523 + 0.0000i
  -0.9523 - 0.0000i
   0.9523 + 0.0000i
  -0.9523 - 0.0000i
   0.9523 + 0.0000i
```

Radiation from 5-Element ULA of Polarized Antennas

Propagate and combine the radiation from five short-dipole antenna elements.

Set up a uniform line array of five short-dipole antennas with polarization enabled. Then, construct the radiator object.

```
antenna = phased.ShortDipoleAntennaElement;
array = phased.ULA('Element',antenna,'NumElements',5);
radiator = phased.Radiator('Sensor',array,'OperatingFrequency',300e6,...
    'CombineRadiatedSignals',true,'Polarization','Combined');
```

Rotate the local coordinate system from the global coordinates by 10° around the x-axis. Demonstrate that the output represents a polarized field.

Specify a simple signal to radiate and specify the radiating direction in azimuth and elevation. Radiate the fields in two directions.

```
x = [1;-1;1;-1;1;-1];
radiatingAngles = [30 30; 0 20];
y = radiator(x,radiatingAngles,rotx(10))
```

```
y=1×2 struct array with fields:
    X
    Y
    Z
```

Show the y-component of the polarized field radiating in the first direction.

```
disp(y(1).Y)
```

```
  -0.2131 + 0.0000i
   0.2131 - 0.0000i
  -0.2131 + 0.0000i
   0.2131 - 0.0000i
  -0.2131 + 0.0000i
   0.2131 - 0.0000i
```

Radiate Signal from Isotropic Antenna

Radiate a signal from a single isotropic antenna.

```
antenna = phased.IsotropicAntennaElement;
radiator = phased.Radiator('Sensor',antenna,'OperatingFrequency',300e6);
sig = [1;1];
radiatingAngles = [30 10]';
y = radiator(sig,radiatingAngles);
```

Radiate a far-field signal in two directions from a 5-element array.

```
array = phased.ULA('NumElements',5);
radiator = phased.Radiator('Sensor',array,'OperatingFrequency',300e6);
sig = [1;1];
radiatingAngles = [30 10; 20 0]';
y = radiator(sig,radiatingAngles);
```

Radiate signals from a 3-element antenna array. Each antenna radiates a separate signal in a separate direction.

```
array = phased.ULA('NumElements',3);
radiator = phased.Radiator('Sensor',array,'OperatingFrequency',1e9,...
    'CombineRadiatedSignals',false);
sig = [1 2 3; 2 8 -1];
radiatingAngles = [10 0; 20 5; 45 2]';
y = radiator(sig,radiatingAngles)
```

```
y = 2×3

     1     2     3
     2     8    -1
```

Measure Target Scattering Matrix Using Dual Polarization

Use a dual-polarization system to obtain target scattering information. Simulate a transmitter and receiver where the vertical and horizontal components are transmitted successively using the input ports of the transmitter. The signals from the two polarization output ports of the receiver are then used to determine the target scattering matrix.

```
scmat = [0 1i; 1i 2];
radiator = phased.Radiator('Sensor', ...
    phased.CustomAntennaElement('SpecifyPolarizationPattern',true), ...
    'Polarization','Dual');
target = phased.RadarTarget('EnablePolarization',true,'ScatteringMatrix', ...
```

```
    scmat);
collector = phased.Collector('Sensor', ...
    phased.CustomAntennaElement('SpecifyPolarizationPattern',true), ...
    'Polarization','Dual');
xh = 1;
xv = 1;
```

Transmit a horizontal component and display the reflected Shh and Svh polarization components.

```
x = radiator(xh,0,[0;0],eye(3));
xrefl = target(x,[0;0],eye(3));
[Shh,Svh] = collector(xrefl,[0;0],eye(3))
```

```
Shh = 0

Svh = 0.0000 + 3.5474i
```

Transmit a vertical component and display the reflected Shv and Svv polarization components.

```
x = radiator(0,xv,[0;0],eye(3));
xrefl = target(x,[0;0],eye(3));
[Shv,Svv] = collector(xrefl,[0;0],eye(3))
```

```
Shv = 0.0000 + 3.5474i

Svv = 7.0947
```

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.WidebandRadiator` | `phased.WidebandCollector` | `phased.Collector` | `phased.FreeSpace` | `twoRayChannel`

step

System object: `phased.Radiator`
Package: `phased`

Radiate signals

Syntax

```
Y = step(H,X,ANG)
Y = step(H,X,ANG,LAXES)
Y = step(H,X,ANG,WEIGHTS)
Y = step(H,X,ANG,STEERANGLE)
Y = step(H,X,ANG,LAXES,WEIGHTS,STEERANGLE)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X,ANG)` radiates signal `X` in the direction `ANG`. `Y` is the radiated signal. The radiating process depends on the `CombineRadiatedSignals` property of `H`, as follows:

- If `CombineRadiatedSignals` has the value `true`, each radiating element or subarray radiates `X` in all the directions in `ANG`. `Y` combines the outputs of all radiating elements or subarrays. If the `Sensor` property of `H` contains subarrays, the radiating process distributes the power equally among the elements of each subarray.
- If `CombineRadiatedSignals` has the value `false`, each radiating element radiates `X` in only one direction in `ANG`. Each column of `Y` contains the output of the corresponding element. The `false` option is available when the `Sensor` property of `H` does not contain subarrays.

`Y = step(H,X,ANG,LAXES)` uses `LAXES` as the local coordinate system axes directions. This syntax is available when you set the `EnablePolarization` property to `true`.

`Y = step(H,X,ANG,WEIGHTS)` uses `WEIGHTS` as the weight vector. This syntax is available when you set the `WeightsInputPort` property to `true`.

`Y = step(H,X,ANG,STEERANGLE)` uses `STEERANGLE` as the subarray steering angle. This syntax is available when you configure `H` so that `H.Sensor` is an array that contains subarrays and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

`Y = step(H,X,ANG,LAXES,WEIGHTS,STEERANGLE)` combines all input arguments. This syntax is available when you configure `H` so that `H.EnablePolarization` is `true`, `H.WeightsInputPort` is `true`, `H.Sensor` is an array that contains subarrays, and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of

the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Radiator object.

X

Signals to radiate. `X` can be either a vector or a matrix.

If `X` is a vector, that vector is radiated through all radiating elements or subarrays. The computation does not divide the signal's power among elements or subarrays, but rather treats the `X` vector the same as a matrix in which each column equals this vector.

If `X` is a matrix, the number of columns of `X` must equal the number of subarrays if `H.Sensor` is an array that contains subarrays, or the number of radiating elements otherwise. Each column of `X` is radiated by the corresponding element or subarray.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

ANG

Radiating directions of signals. `ANG` is a two-row matrix. Each column specifies a radiating direction in the form `[AzimuthAngle;ElevationAngle]`, in degrees.

LAXES

Local coordinate system. `LAXES` is a 3-by-3 matrix whose columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively. Each axis is specified in terms of `[x;y;z]` with respect to the global coordinate system. This argument is only used when the `EnablePolarization` property is set to `true`.

WEIGHTS

Vector of weights. `WEIGHTS` is a column vector whose length equals the number of radiating elements or subarrays.

STEERANGLE

Subarray steering angle, specified as a length-2 column vector. The vector has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90° and 90°, inclusive.

Output Arguments

Y

Radiated signals

- If the `EnablePolarization` property value is set to `false`, the output argument `Y` is a matrix. The number of columns of the matrix equals the number of radiating signals. Each column of `Y` contains a separate radiating signal. The number of radiating signals depends upon the `CombineRadiatedSignals` property of `H`.
- If the `EnablePolarization` property value is set to `true`, `Y` is a row vector of elements of MATLAB `struct` type. The length of the `struct` vector equals the number of radiating signals. Each `struct` contains a separate radiating signal. The number of radiating signals depends upon the `CombineRadiatedSignals` property of `H`. Each `struct` contains three column-vector fields, `X`, `Y`, and `Z`. These fields represent the x, y, and z components of the polarized wave vector signal in the global coordinate system.

Examples

Radiation from 5-Element ULA

Propagate and combine radiation from five isotropic antenna elements. Set up a uniform line array of five isotropic antennas.

First construct a ULA array.

```
array = phased.ULA('NumElements',5);
```

Construct a radiator object.

```
radiator = phased.Radiator('Sensor',array,...
    'OperatingFrequency',300e6,'CombineRadiatedSignals',true);
```

Create a simple signal to radiate.

```
x = [1;-1;1;-1;1;-1];
```

Specify the azimuth and elevation of the radiating direction.

```
radiatingAngle = [30;10];
```

Radiate the signal.

```
y = radiator(x,radiatingAngle)
```

```
y = 6×1 complex

  -0.9523 - 0.0000i
   0.9523 + 0.0000i
  -0.9523 - 0.0000i
   0.9523 + 0.0000i
  -0.9523 - 0.0000i
   0.9523 + 0.0000i
```

Radiation from 5-Element ULA of Polarized Antennas

Propagate and combine the radiation from five short-dipole antenna elements.

Set up a uniform line array of five short-dipole antennas with polarization enabled. Then, construct the radiator object.

```
antenna = phased.ShortDipoleAntennaElement;
array = phased.ULA('Element',antenna,'NumElements',5);
radiator = phased.Radiator('Sensor',array,'OperatingFrequency',300e6,...
    'CombineRadiatedSignals',true,'Polarization','Combined');
```

Rotate the local coordinate system from the global coordinates by 10° around the x-axis. Demonstrate that the output represents a polarized field.

Specify a simple signal to radiate and specify the radiating direction in azimuth and elevation. Radiate the fields in two directions.

```
x = [1;-1;1;-1;1;-1];
radiatingAngles = [30 30; 0 20];
y = radiator(x,radiatingAngles,rotx(10))
```

```
y=1×2 struct array with fields:
    X
    Y
    Z
```

Show the y-component of the polarized field radiating in the first direction.

```
disp(y(1).Y)
```

```
  -0.2131 + 0.0000i
   0.2131 - 0.0000i
  -0.2131 + 0.0000i
   0.2131 - 0.0000i
  -0.2131 + 0.0000i
   0.2131 - 0.0000i
```

phased.RangeAngleResponse

Package: `phased`

Range-angle response

Description

The `phased.RangeAngleResponse` System object creates an range-angle response object. This object calculate the range-angle response of a signal using either a matched filter or an FFT.

The input to the range-angle response object is a data cube. The organization of the data cube follows the Phased Array System Toolbox convention. The first dimension of the cube represents the fast-time samples or ranges of the received signals. The second dimension represents multiple channels such as sensors or beams. The third dimension, slow time, represents pulses or sweeps. If the data contains only one channel, for example, the data cube can contain fewer than three dimensions. Range processing operates along the first dimension of the cube. Angle processing operates along the second dimension.

The output of the object is also a data cube with the same number of dimensions as the input. The first dimension contains range-processed data but its length can differ from the first dimension of the input. The second dimension contains angle-processed data. Its length can differ from the last dimension of the input.

To obtain the range-angle response:

1. Create the `phased.RangeAngleResponse` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
response = phased.RangeAngleResponse
response = phased.RangeAngleResponse(Name,Value)
```

Description

`response = phased.RangeAngleResponse` creates a `phased.RangeAngleResponse` System object, `response`, with default property values.

`response = phased.RangeAngleResponse(Name,Value)` sets properties for the `phased.RangeAngleResponse` object using one or more name-value pairs. For example, `response = phased.RangeAngleResponse('RangeMethod','FFT','SampleRate',1e6)` creates an object that uses an FFT range processing method at a sample rate of 1 MHz. Enclose property names in quotes.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SensorArray` — Sensor array
phased.ULA array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object.

Example: `phased.URA`

`RangeMethod` — Range processing method
`'Matched filter'` (default) | `'FFT'`

Range processing method, specified as `'Matched filter'` or `'FFT'`.

- `'Matched filter'` — The object match-filters the incoming signal. This approach is commonly used for pulsed signals, where the matched filter is the time reverse of the transmitted signal.
- `'FFT'` — The object applies an FFT to the input signal. This approach is commonly used for chirped signals such as FMCW and linear FM pulsed signals.

Example: `'Matched filter'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `double`

`SampleRate` — Signal sample rate
`1e6` (default) | positive real-valued scalar

Signal sample rate, specified as a positive real-valued scalar. Units are in hertz.

Example: `1e6`

Data Types: `double`

`SweepSlope` — Linear FM sweep slope
`1.0e9` (default) | scalar

Linear FM sweep slope, specified as a scalar. The fast-time dimension of the `signal` input argument to `step` must correspond to sweeps having this slope.

Example: `1.5e9`

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'`.

Data Types: `double`

`DechirpInput` — Enable dechirping of input signals
`false` (default) | `true`

Option to enable dechirping of input signals, specified as `false` or `true`. Set this property to `false` to indicate that the input signal is already dechirped and no dechirp operation is necessary. Set this property to `true` when the input signal requires dechirping.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'`.

Data Types: `logical`

`DecimationFactor` — Decimation factor for dechirped signals
`1` (default) | positive integer

Decimation factor for dechirped signals, specified as a positive integer. The decimation algorithm uses a 30th-order FIR filter generated by `fir1(30,1/D)`, where `D` is the decimation factor. The default value of `1` implies no decimation.

When processing FMCW signals, decimating the dechirped signal is useful for reducing the load on A/D converters.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `true`.

Data Types: `double`

`RangeFFTLengthSource` — Source of FFT length for range processing of dechirped signals
`'Auto` (default) | `'Property'`

Source of the FFT length used for the range processing of dechirped signals, specified as `'Auto'` or `'Property'`.

- `'Auto'` — The FFT length equals the length of the fast-time dimension of the input data cube.
- `'Property'` — Specify the FFT length by using the `RangeFFTLength` property.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'`.

Data Types: `char`

`RangeFFTLength` — FFT length used for range processing
`1024` (default) | positive integer

FFT length used for range processing, specified as a positive integer.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeFFTLengthSource` property to `'Property'`

Data Types: `double`

`RangeWindow` — FFT weighting window for range processing
`'None'` (default) | `'Hamming'` | `'Chebyshev'` | `'Hann'` | `'Kaiser'` | `'Taylor'` | `'Custom'`

FFT weighting window for range processing, specified as `'None'`, `'Hamming'`, `'Chebyshev'`, `'Hann'`, `'Kaiser'`, `'Taylor'`, or `'Custom'`.

If you set this property to `'Taylor'`, the generated Taylor window has four nearly constant sidelobes next to the mainlobe.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'`.

Data Types: `char`

`RangeSidelobeAttenuation` — Sidelobe attenuation for range processing
`30` (default) | positive scalar

Sidelobe attenuation for range processing, specified as a positive scalar. Attenuation applies to Kaiser, Chebyshev, or Taylor windows. Units are in dB.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeWindow` property to `'Kaiser'`, `'Chebyshev'`, or `'Taylor'`.

`CustomRangeWindow` — Custom window for range processing
`@hamming` (default) | function handle | cell array

Custom window for range processing, specified as a function handle or a cell array containing a function handle as its first entry. If you do not specify a window length, the object computes the window length and passes that into the function. If you specify a cell array, the remaining cells of the array can contain arguments to the function. If you use only the function handle without passing in arguments, all arguments take their default values.

If you write your own window function, the first argument must be the length of the window.

Note Instead of using a cell array, you can pass in all arguments by constructing a handle to an anonymous function. For example, you can set the value of `CustomRangeWindow` to `@(n)taylorwin(n,nbar,sll)`, where you have previously set the values of `nbar` and `sll`.

Example: `{@taylor,5,-35}`

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeWindow` property to `'Custom'`.

Data Types: `function_handle` | `cell`

`ReferenceRangeCentered` — Set reference range at center of range grid
`true` (default) | `false`

Set reference range at center of range grid, specified as `true` or `false`. Setting this property to `true` enables you to set the reference range at the center of the range grid. Setting this property to `false` sets the reference range to the beginning of the range grid.

Dependencies

To enable this property, set the `RangeMethod` to `'FFT'`.

Data Types: `logical`

`ReferenceRange` — Reference range of range grid
`0.0` (default) | nonnegative scalar

Reference range of the range grid, specified as a nonnegative scalar.

- If you set the `RangeMethod` property to `'Matched filter'`, the reference range is set to the start of the range grid.
- If you set the `RangeMethod` property to `'FFT'`, the reference range is determined by the `ReferenceRangeCentered` property.

  - When you set the `ReferenceRangeCentered` property to `true`, the reference range is set to the center of the range grid.
  - When you set the `ReferenceRangeCentered` property to `false`, the reference range is set to the start of the range grid.

  Units are in meters.

This property is tunable.

Example: `1000.0`

Data Types: `double`

`MaximumNumInputSamplesSource` — Source of maximum number of input signal samples
`'Auto'` (default) | `'Property'`

Source of the maximum number of input signal samples, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the object automatically allocates enough memory to buffer the first input signal. When you set this property to `'Property'`, you specify the maximum number of samples in the input signal using the `MaximumNumInputSamples` property. Any input signal longer than that value is truncated.

To use this object with a variable-size signal in a MATLAB Function Block in Simulink, set this property to `'Property'` and set a value for the `MaximumNumInputSamples` property.

Dependencies

To enable this property, set the `MaximumDistanceSource` property to `'Property'`.

MaximumNumInputSamples — Maximum number of input signal samples
100 (default) | positive integer

Maximum number of samples in the input signal, specified as a positive integer. This property limits the size of the input signal. The input signal is the first argument to the object. The number of samples is the number of rows in the input. An input signal longer than this value is truncated.

Example: 1024

Dependencies

To enable this property, set the RangeMethod property to 'Matched filter' and set the MaximumNumInputSamplesSource property to 'Property'.

Data Types: double

ElevationAngleSource — Source of elevation angle
'Property' (default) | 'Input port'

Source of elevation angle, specified as 'Property' or 'Input port'.

'Property'	The elevation angle comes from the ElevationAngle property.
'Input port'	The elevation angle comes from an input argument.

ElevationAngle — Elevation angle
0 (default) | scalar

Specify the elevation angle in degrees used to calculate the range-angle response as a scalar. The angle must lie in the range from –90° to 90°. Units are in degrees.

Example: 45.0

Dependencies

To enable this property, set the ElevationAngleSource property to 'Property'.

Data Types: double

AngleSpan — Angle response span
[-90 90] (default) | real-valued 1-by-2 vector

Angle response span, specified as a real-valued 2-by-1 vector. The object calculates the range-angle response within the angle range, [min_angle max_angle].

Example: [-45 45]

Data Types: double

NumAngleSamples — Number of samples in angle span
positive integer greater than two

Number of samples in angle span used to calculate range-angle response, specified as a positive integer greater than two.

Example: [256]

Data Types: double

You can combine optional input arguments when their enabling properties are set. Optional inputs must be listed in the same order as the order of the enabling properties. For example,

```
[RESP,RANGE,ANG] = response(X,XREF,EL)
```

or

```
[RESP,RANGE,ANG] = response(X,COEFF,EL)
```

Usage

Syntax

```
[RESP,RANGE,ANG] = response(X)
[RESP,RANGE,ANG] = response(X,XREF)
[RESP,RANGE,ANG] = response(X,COEFF)
[RESP,RANGE,ANG] = response( ___ ,EL)
```

Description

`[RESP,RANGE,ANG] = response(X)` returns the range-angle response, `RESP`, the ranges, `RANGE`, and the angles, `ANG`. `X` is a dechirped signal. This syntax applies when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `false`. This syntax is often applied to FMCW signals.

`[RESP,RANGE,ANG] = response(X,XREF)` also specifies the reference signal, `XREF` to dechirped the signal. This syntax applies when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `true`. This syntax is often applied to FMCW signals. Then, the reference signal can be the transmitted signal.

`[RESP,RANGE,ANG] = response(X,COEFF)` also specifies `COEFF` as matched filter coefficients. This syntax applies when you set the `RangeMethod` property to `'MatchedFilter'`. This syntax is often applied to pulsed signals.

`[RESP,RANGE,ANG] = response( ___ ,EL)` also specifies `EL` as the elevation angle. This syntax applies when you set the `ElevationAngleSource` property to `'Input port'`.

Input Arguments

X — Input signal data cube

complex-valued *K*-by-*N* matrix | complex-valued *K*-by-*N*-by-*L* array

Input signal cube, specified as a complex-valued *K*-by-*N* matrix or complex-valued *K*-by-*N*-by-*L* array. The contents of the data cube depend on the type of range-angle processing specified by the different syntaxes.

- *K* is the number of fast-time or range samples.
- *N* is the number of independent spatial channels such as sensors or beams.
- *L* is the slow-time dimension that corresponds to the number of pulses or sweeps in the input signal.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

XREF — Reference signal used for dechirping

complex-valued *K*-by-1 column vector

Reference signal used for dechirping, specified as a complex-valued *K*-by-1 column vector. The number of rows must equal the length of the fast-time dimension of `X`.

Dependencies

To enable this input argument, set the value of `RangeMethod` to `'FFT'` and `DechirpInput` to `true`.

Data Types: `double`

COEFF — Matched filter coefficients

complex-valued *P*-by-1 column vector

Matched filter coefficients, specified as a complex-valued *P*-by-1 column vector. *P* must be less than or equal to *K*. *K* is the number of fast-time or range sample.

Dependencies

To enable this input argument, set the value of `RangeMethod` to `'Matched filter'`.

Data Types: `double`

EL — Elevation angle

scalar

Elevation angle of response, specified as a scalar between –90° and +90°. The range-angle response is computed for this elevation. Units are in degrees.

Dependencies

To enable this argument, set the `ElevationAngleSource` property to `'Input port'`.

Data Types: `double`

Output Arguments

RESP — Range response data cube

complex-valued *M*-element column vector | complex-valued *M*-by-*L* matrix | complex-valued *M*-by-*N* by-*L* array

Range response data cube, returned as one of the following:

- Complex-valued *M*-element column vector
- Complex-valued *M*-by-*L* matrix
- Complex-valued *M*-by-*N* by-*L* array

The value of *M* depends on the type of processing

RangeMethod Property	DechirpInput Property	Value of *M*
'FFT'	false	If you set the RangeFFTLength property to 'Auto', *M* = *K*, the length of the fast-time dimension of x. Otherwise, *M* equals the value of the RangeFFTLength property.
	true	*M* equals the quotient of the number of rows, *K*, of the input signal by the value of the decimation factor, *D*, specified in DecimationFactor.
'Matched filter'	n/a	*M* = *K*, the length of the fast-time dimension of x.

Data Types: double

RANGE — Range values along range dimension
real-valued *M*-by-1 column vector

Range values along range dimension, returned as a real-valued *M*-by-1 column vector. rnggrid defines the ranges corresponding to the fast-time dimension of the RESP output data cube. *M* is the length of the fast-time dimension of RESP. Range values are monotonically increasing and equally spaced. Units are in meters.

Data Types: double

ANG — Angle values along angle direction
P-by-1 real-valued vector

Angle values along angle direction, returned as a *P*-by-1 real-valued vector. Units are in degrees.

Data Types: double

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named obj, use this syntax:

```
release(obj)
```

Specific to phased.RangeAngleResponse

plotResponse	Plot range-angle response

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Range Angle Response of Antenna Array

Calculate the range-angle response from a pulsed radar transmitting a rectangular waveform using the matched filter approach. The signal includes three target returns. Two are approximately 2000 m away and the third is approximately 3500 m away. In addition, two targets are stationary relative to the radar while the third is moving away from the radar at approximately 100 m/s. The signals arrive at an 8-element uniform linear array.

First, load the example data.

```
load('RangeAngleResponseExampleData','rectdata');
fs = rectdata.fs;
propspeed = rectdata.propspeed;
fc = rectdata.fc;
rxdata = rectdata.rxdata;
mfcoeffs = rectdata.mfcoeffs;
%noisepower = rectdata.noisepower;
antennaarray = rectdata.antennaarray;
```

Second, create the range-angle response object for matched filter processing.

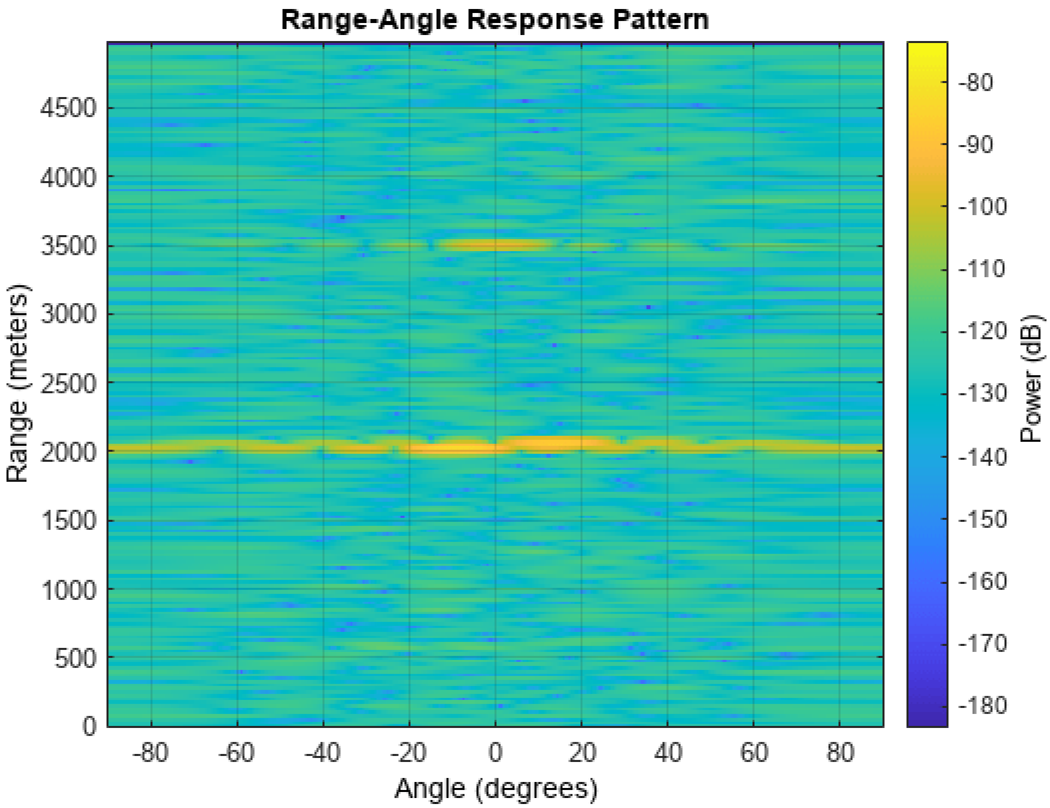
```
rngangresp = phased.RangeAngleResponse(...
    'SensorArray',antennaarray,'OperatingFrequency',fc,...
    'SampleRate',fs,'PropagationSpeed',propspeed);
```

Obtain the range-angle map.

```
[resp,rng_grid,ang_grid] = rngangresp(rxdata,mfcoeffs);
```

Plot the response.

```
plotResponse(rngangresp,rxdata,mfcoeffs,'Unit','db');
```



Algorithms

Range-Angle Response

The object generates the response by first processing the input signal in the range domain using either a matched filter or a dechirp operation and then by processing along azimuth angles.

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `CustomRangeWindow` property is not supported.
- The `plotResponse` object function is not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Functions
`bw2range` | `fir1` | `chebwin` | `dechirp` | `hann` | `hamming` | `kaiser` | `taylorwin` | `rangeangle`

Objects
`phased.RangeResponse` | `phased.RangeDopplerResponse` | `phased.AngleDopplerResponse` | `phased.MatchedFilter` | `phased.DopplerEstimator` | `phased.RangeEstimator` | `phased.CFARDetector` | `phased.CFARDetector2D`

plotResponse

Package: `phased`

Plot range-angle response

Syntax

```
plotResponse(response,X)
plotResponse(response,X,XREF)
plotResponse(response,X,COEFF)
plotResponse( ___ ,'Unit',unit)
```

Description

`plotResponse(response,X)` plots the range response of a dechirped input signal, `X`, from the `phased.RangeAngleResponse` object, `response`. This syntax applies when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `false`.

`plotResponse(response,X,XREF)` plots the range response `X`, after performing a dechirp operation using the reference signal, `XREF`. This syntax applies when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `true`.

`plotResponse(response,X,COEFF)` plots the range response of `X` after match filtering using the match filter coefficients, `coeff`. This syntax applies when you set the `RangeMethod` property to `'Matched filter'`.

`plotResponse( ___ ,'Unit',unit)` plots the response in the units specified by `units`.

Examples

Range Angle Response of Antenna Array

Calculate the range-angle response from a pulsed radar transmitting a rectangular waveform using the matched filter approach. The signal includes three target returns. Two are approximately 2000 m away and the third is approximately 3500 m away. In addition, two targets are stationary relative to the radar while the third is moving away from the radar at approximately 100 m/s. The signals arrive at an 8-element uniform linear array.

First, load the example data.

```
load('RangeAngleResponseExampleData','rectdata');
fs = rectdata.fs;
propspeed = rectdata.propspeed;
fc = rectdata.fc;
rxdata = rectdata.rxdata;
mfcoeffs = rectdata.mfcoeffs;
%noisepower = rectdata.noisepower;
antennaarray = rectdata.antennaarray;
```

Second, create the range-angle response object for matched filter processing.

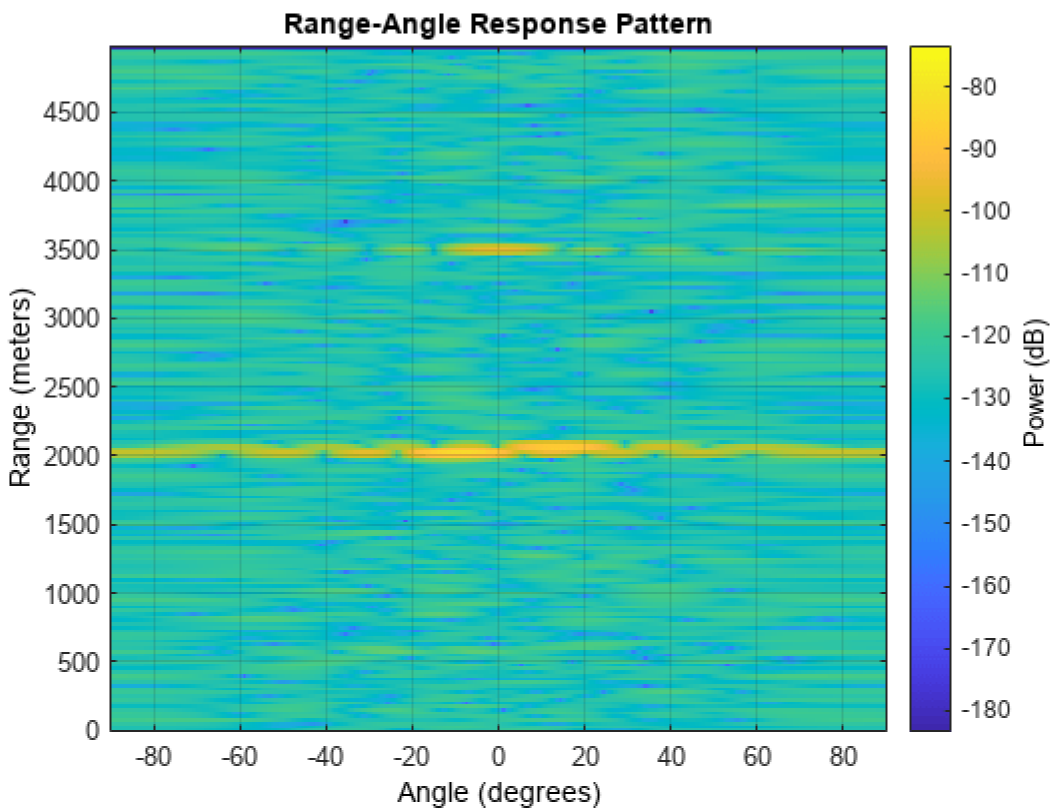
```
rngangresp = phased.RangeAngleResponse(...
    'SensorArray',antennaarray,'OperatingFrequency',fc,...
    'SampleRate',fs,'PropagationSpeed',propspeed);
```

Obtain the range-angle map.

```
[resp,rng_grid,ang_grid] = rngangresp(rxdata,mfcoeffs);
```

Plot the response.

```
plotResponse(rngangresp,rxdata,mfcoeffs,'Unit','db');
```



Input Arguments

response — Range-angle response object
phased.RangeAngleResponse System object

Range-angle response object, specified as a `phased.RangeAngleResponse` System object.

X — Input data
complex-valued *K*-by-*N* matrix

Input data, specified as a complex-valued *K*-by-*N* matrix. The contents of the data cube depend on the type of range-angle processing specified by the different syntaxes. *K* always specifies the number of fast-time samples and *N* is always the number of channels, either array elements or beams.

- *K* is the number of fast-time or range samples.
- *N* is the number of independent spatial channels such as sensors or directions.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

XREF — Reference signal used for dechirping
complex-valued *K*-by-1 column vector

Reference signal used for dechirping, specified as a complex-valued *K*-by-1 column vector. The number of rows must equal the length of the fast-time dimension of `X`.

Dependencies

To enable this input argument, set the value of `RangeMethod` to `'FFT'` and `DechirpInput` to `true`.

Data Types: `double`

COEFF — Matched filter coefficients
complex-valued *P*-by-1 column vector

Matched filter coefficients, specified as a complex-valued *P*-by-1 column vector. *P* must be less than or equal to *K*. *K* is the number of fast-time or range sample.

Dependencies

To enable this input argument, set the value of `RangeMethod` to `'Matched filter'`.

Data Types: `double`

unit — Plot units
`'db'` (default) | `'mag'` | `'pow'`

Plot units, specified as `'db'`, `'mag'`, or `'pow'`. who

- `'db'` – plot the response power in dB.
- `'mag'` – plot the magnitude of the response.
- `'pow'` – plot the response power.

Example: `'mag'`

Data Types: `char` | `string`

Version History

Introduced in R2018b

phased.RangeDopplerScope

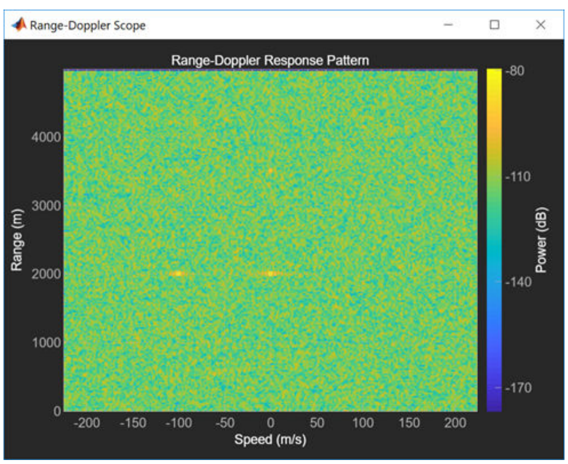
Package: `phased`

Range-Doppler scope

Description

The `phased.RangeDopplerScope` System object creates a scope for viewing a range- response map. The map is a 2-D image of response intensity as a function of range and (or speed). You can input two types of data - in-phase and quadrature (I/Q) data and response data.

- I/Q data – The data consists of fast-time and slow-time I/Q samples of pulses or sweeps. The scope computes and displays the response map. To use I/Q data, set the `IQDataInput` property to `true`. In this mode, you can set the properties shown in “Properties Applicable to I/Q Data” on page 1-1307.
- Response data – The data consists of the range- response itself. The scope displays the range-response map. For example, you can obtain range- response from `phased.RangeDopplerResponse` object. To use response data, set the `IQDataInput` property to `false`. In this mode, you can set the properties shown in “Properties Applicable to Response Data” on page 1-1308.



To display a range-Doppler response map using the scope,

1. Create the `phased.RangeDopplerScope` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
scope = phased.RangeDopplerScope
```

```
scope = phased.phased.RangeDopplerScope(Name,Value)
```

Description

`scope = phased.RangeDopplerScope` creates a range-Doppler scope System object, `scope`. This object displays the range-Doppler response of the input data.

`scope = phased.phased.RangeDopplerScope(Name,Value)` creates a range-Doppler scope object, `scope`, with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose property names in quotes. For example,

```
scope = phased.RangeDopplerScope('IQInputData',true,'RangeMethod', ...
        'FFT','SampleRate',1e6,'DopplerOutput','Speed', ...
        'OperatingFrequency',10e6,'SpeedUnits','km/h');
```

creates a scope object that uses FFT-based range processing for I/Q data having a sample rate of 1 MHz. The Doppler output units are speed in kilometers per hour.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

Name — Display caption
`'Range-Doppler Scope'` (default) | character vector

Display caption, specified as a character vector. The caption appears in the title bar of the window.

Example: `'Aircraft Range-Doppler Response'`

Tunable: Yes

Data Types: `char`

Position — Location and size of intensity scope window
depends on display-resolution (default) | 1-by-4 vector of positive values

Location and size of the intensity scope window, specified as a 1-by-4 vector having the form `[left bottom width height]`.

- `left` and `bottom` specify the location of the bottom-left corner of the window.
- `width` and `height` specify the width and height of the window.

Units are in pixels.

The default value of this property depends on the resolution of your display. By default, the window is positioned in the center of the screen, with a width and height of 800 and 450 pixels, respectively.

Example: `[100 100 500 400]`

Tunable: Yes

Data Types: `double`

`IQDataInput` — Type of input data
`true` (default) | `false`

Type of input data, specified as `true` or `false`. When `true`, the object assumes that the input consists of I/Q sample data and further processing is required in the range and Doppler domains. When `false`, the object assumes that the data is response data that has already been processed.

Data Types: `logical`

`ResponseUnits` — Response units label
`'db'` (default) | `'magnitude'` | `'power'`

Response units, specified as `'db'`, `'magnitude'`, or `'power'`.

Data Types: `char`

`RangeLabel` — Range-axis label
`'Range (m)'` (default) | character vector

Range-axis label, specified as a character vector.

Example: `'Range (km)'`

Tunable: Yes

Dependencies

To enable this property, set the `IQDataInput` to `false`.

Data Types: `char`

`DopplerLabel` — Doppler-axis label
`'Doppler Frequency (Hz)'` (default) | character vector

Doppler-axis label, specified as a character vector.

Example: `'Doppler Frequency (kHz)'`

Tunable: Yes

Dependencies

To enable this property, set the `IQDataInput` to `false`.

Data Types: `char`

`RangeMethod` — Range processing method
`'Matched filter'` (default) | `'FFT'`

Range-processing method, specified as `'Matched filter'` or `'FFT'`.

`'Matched filter'`	The object applies a matched filter to the incoming signal. This approach is commonly used with pulsed signals, where the matched filter is a time-reversed replica of the transmitted signal.

'FFT'	Algorithm performs range processing by applying an FFT to the input signal. This approach is commonly used with FMCW continuous signals and linear FM pulsed signals.

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

RangeUnits — Range units

'm' (default) | 'km' | 'mi' | 'nmi'

Range units, specified as:

- 'm' – meters
- 'km' – kilometers
- 'mi' – miles
- 'nmi' – nautical miles

Example: 'mi'

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `char`

PropagationSpeed — Signal propagation speed

`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value of this property is the speed of light. See `physconst`. Units are in meters/second.

Example: `3e8`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

SampleRate — Sample rate

`1e6` (default) | positive scalar

Sample rate, specified as a positive scalar. Units are in Hz.

Example: `10e3`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

SweepSlope — FM sweep slope

`1e9` (default) | scalar

Slope of the linear FM sweep, specified as a scalar. Units are in Hz/sec.

Dependencies

To enable this property, set the IQDataInput property to true and the RangeMethod property to 'FFT'.

Data Types: double

DechirpInput — Dechirp input signal
false (default) | true

Set this property to true to dechirp the input signal before performing range processing. false indicates that the input signal is already dechirped and no dechirp operation is necessary.

Dependencies

To enable this property, set the IQDataInput property to true and the RangeMethod property to 'FFT'.

Data Types: logical

RangeFFTLength — FFT length used in range processing
1024 (default) | positive integer

FFT length used for range processing, specified as a positive integer.

Example: 128

Dependencies

To enable this property, set the IQDataInput property to true and the RangeMethod property to 'FFT'.

Data Types: double

ReferenceRangeCentered — Set reference range at center of range span
true (default) | false

Set this property to true to set the reference range to the center of the range span. Set this property to false to set the reference range to the beginning of the range span.

Dependencies

To enable this property, set the IQDataInput property to true and the RangeMethod property to 'FFT'.

Data Types: logical

ReferenceRange — Reference range
0.0 (default) | nonnegative scalar

Reference range of the range span, specified as a nonnegative scalar.

- If you set the RangeMethod property to 'Matched filter', the reference range marks the start of the range span.
- If you set the RangeMethod property to 'FFT', the position of the reference range depends on the ReferenceRangeCentered property.
  - If you set the ReferenceRangeCentered property to true, the reference range marks the center of the range span.

- If you set the ReferenceRangeCentered property to false, the reference range marks the start of the range span.

Units are in meters.

Example: 1000.0

Tunable: Yes

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: double

PRFSource — Source of pulse repetition frequency
'Auto' (default) | 'Property'

Source of the pulse repetition frequency (PRF) of the input signal, specified as 'Auto' or 'Property'. When you set this property to 'Auto', the PRF is a function of the number of rows in the input signal and the value of the SampleRate property. When you set this property to 'Property', you can specify the PRF using the PRF property.

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: char

PRF — Pulse repetition frequency of input signal
10e3 (default) | positive scalar

Pulse repetition frequency of input signal, specified as a positive scalar. Units are in Hz.

Example: 1.4e3

Dependencies

To enable this property, set the IQDataInput property to true and set the PRFSource property to 'Property'.

Data Types: double

DopplerFFTLength — FFT length used in Doppler processing
1024 (default) | positive integer

FFT length used in Doppler processing, specified as a positive integer.

Example: 67

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: double

DopplerOutput — Doppler output
'Frequency' (default) | 'Speed'

Doppler output, specified as `'Frequency'` or `'Speed'`. If you set this property to `'Frequency'`, the Doppler output, `Dop`, at object execution time is the Doppler shift. If you set this property to `'Speed'`, the Doppler output is the equivalent radial speed.

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `char`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar.

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `DopplerOutput` to `'Speed'`.

Data Types: `double`

`NormalizeDoppler` — Normalize Doppler
`false` (default) | `true`

Set this property to `true` to plot the range-Doppler response with normalized Doppler frequency. Set this property to `false` to plot the range-Doppler response without normalizing the Doppler frequency.

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `DopplerOutput` to `'Frequency'`.

Data Types: `logical`

`SpeedUnits` — Doppler speed units
`'m/s'` (default) | `'km/h'` | `'mph'` | `'kt'`

Doppler speed units:

- `'m/s'` – meters per second
- `'km/h'` – kilometers per hour
- `'mph'` – miles per hour
- `'kt'` – knots or nautical miles per hour

Example: `'mph'`

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `DopplerOutput` property to `'Speed'`.

Data Types: `char`

`FrequencyUnits` — Doppler frequency units
`'Hz'` (default) | `'kHz'` | `'MHz'`

Doppler frequency units, specified as `'Hz'`, `'kHz'`, or `'MHz'`.

Example: `'MHz'`

Dependencies

To enable this property, set the `IQDataInput` property to `true`, the `DopplerOutput` to `'Frequency'`, and the `NormalizedDoppler` property to `false`.

Data Types: `char`

Usage

Syntax

```
scope(X,Range,Dop)
```

```
scope(X)
scope(X,XREF)
scope(X,COEFF)
```

Description

`scope(X,Range,Dop)` displays a range-Doppler response map, `X`, at the ranges, `Range`, and Doppler shifts, `Dop`. This syntax applies when you set the `IQDataInput` to `false`.

`scope(X)` computes and displays the range-Doppler response map. This syntax applies when you set the `IQDataInput` property to `true`, the `RangeMethod` property to `'FFT'`, and the `DechirpInput` property to `false`. This syntax is most commonly used with FMCW signals. All sweeps in `X` are assumed to be contiguous. If the sweeps are not contiguous, set the PRF by setting the `PRFSource` property to `'Property'` and the `PRF` of the input data to the PRF.

`scope(X,XREF)` also specifies a reference signal to use for dechirping the input signal, `X`. This syntax applies when you set the `IQDataInput` property to `true`, the `RangeMethod` property to `'FFT'`, and the `DechirpInput` property to `true`. This syntax is most commonly used with FMCW signals. `XREF` is generally the transmitted signal.

`scope(X,COEFF)` also specifies matched filter coefficients, `COEFF`. This syntax applies when you set the `IQDataInput` property to `true` and the `RangeMethod` property to `'Matched Filter'`. This syntax is most commonly used with pulsed signals.

Input Arguments

X — Input data
complex-valued *K*-by-*L* matrix

Input data, specified as a complex-valued *K*-by-*L* matrix. The interpretation of the data depends on the value of the `IQDataInput` property.

- When `IQDataInput` is `true`, the input consists of received fast-time (range) samples for each PRI pulse or FMCW sweep. *K* denotes the number of fast-time samples. *L* is the number of Doppler samples. The number of Doppler samples is the number of pulses in the case of pulsed signals or the number of dechirped frequency sweeps for FMCW signals. The scope computes and displays the range-Doppler response.

  - When RangeMethod is set to 'FFT' and DechirpInput is false, X has previously been dechirped.
  - When RangeMethod is set to 'FFT' and DechirpInput is true, X has not been previously dechirped. Use the syntax that includes XREF as input data.
  - When RangeMethod is set to 'MatchedFilter', X has not been match filtered. Use the syntax that includes COEFF as input data.
- When IQDataInput is false, the input already consists of response data in the range-Doppler domain such as that produced by phased.RangeDopplerResponse. Each row of the response map corresponds to an element of the Range vector. Each column corresponds to an element of the Dop vector. The scope serves only as a display of the range-Doppler response.

Range — Range grid values of range-Doppler response map
real-valued *K*-by-1 column vector

Range grid values of response map, specified as a real-valued *K*-by-1 column vector. Range denotes the range values at which the response has been computed. Elements of Range correspond to the rows of X.

Dependencies

To enable this argument, set the IQInputData property to false.

Data Types: double

Dop — Doppler grid values of range-Doppler response map
real-valued *L*-by-1 column vector

Doppler grid values of response map, specified as a real-valued *L*-by-1 column vector. Dop denotes the Doppler values at which the response has been computed. Elements of Dop correspond to the columns of X. Dop can contain either Doppler or speed values at which the range-Doppler response is evaluated.

Dependencies

To enable this argument, set the IQInputData property to false.

Data Types: double

XREF — Reference signal
complex-valued *K*-by-1 column vector

Reference signal used to dechirp X. XREF must be a column vector with the same number of rows as X.

Dependencies

To enable this argument, set the IQDataInput property to true, the RangeMethod property to 'FFT' and the DechirpInput property to false

Data Types: double
Complex Number Support: Yes

COEFF — Matched filter coefficients
complex-valued column vector

Matched filter coefficients, specified as a complex-valued column vector.

Dependencies

To enable this argument, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'Matched Filter'`.

Data Types: `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Scope Objects

show	Turn on visibility of scopes
hide	Turn off visibility of scope
isVisible	Visibility of scopes

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

View Target Response Using Range-Doppler Scope

Calculate and visualize the range-Doppler response from a pulsed radar transmitting a rectangular waveform. Compute the response using matched filtering. The signal contains returns from three targets. One target is approximately 2000 m away and is stationary relative to the radar. The second target is approximately 3500 m away and is also stationary relative to the radar. The third is approximately 2000 m away and is moving away from the radar at approximately 100 m/s.

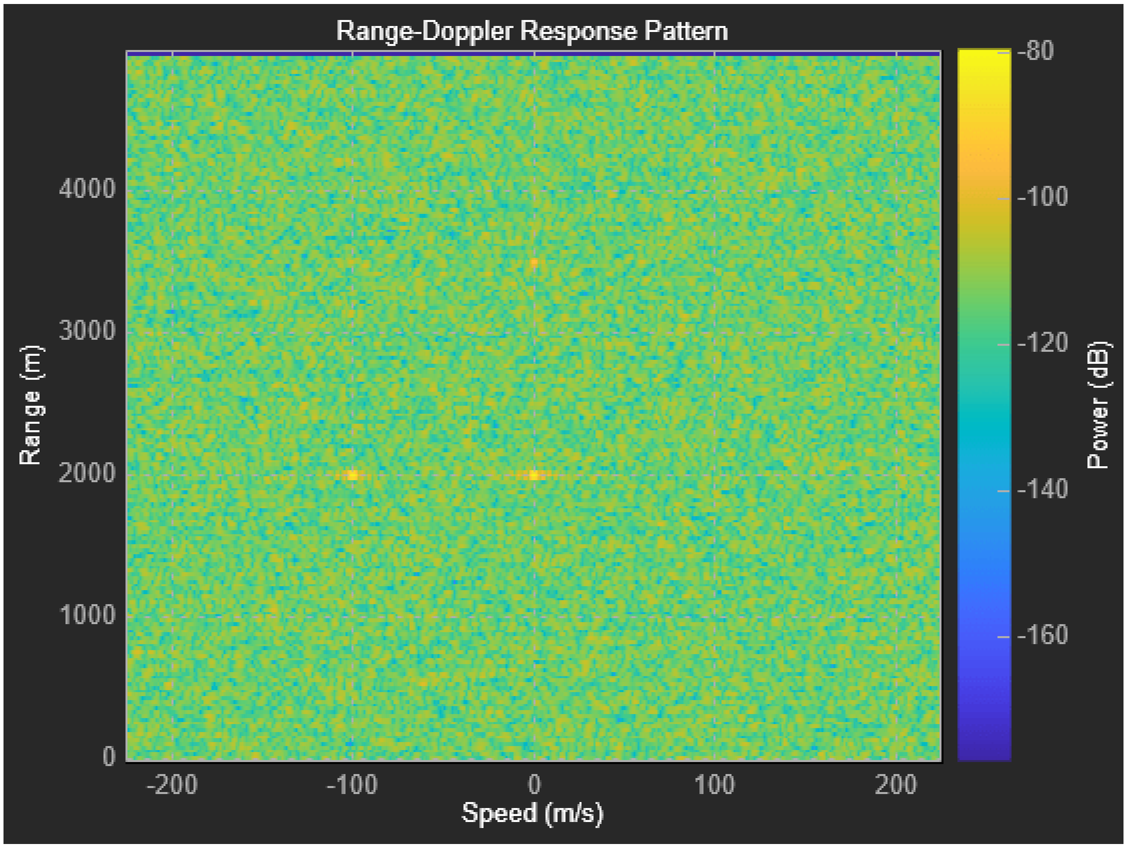
Load the IQ data and obtain the signals and parameters.

```
load('RangeDopplerResponseExampleData','rectdata');
fs = rectdata.fs;
c = rectdata.propspeed;
fc = rectdata.fc;
rxdata = rectdata.rxdata;
mfcoeffs = rectdata.mfcoeffs;
```

Create the range-Doppler scope for matched filter processing and visualization. Set the Doppler FFT size to 1024. The display shows the three targets.

```
scope = phased.RangeDopplerScope( ...
    'IQDataInput',true,'RangeMethod','Matched filter', ...
    'Name','Range-Doppler Scope', ...
    'Position',[560 375 560 420],'ResponseUnits','db', ...
```

```
    'RangeUnits','m','DopplerFFTLength',1024, ...
    'DopplerOutput','Speed','OperatingFrequency',fc, ...
    'SampleRate',fs,'PropagationSpeed',c);
scope(rxdata,mfcoeffs);
```



More About

Properties Applicable to I/Q Data

These properties are applicable when `IQDataInput` is `true`.

Properties	
`Name`	`Position`
`ResponseUnits`	`RangeMethod`
`RangeUnits`	`PropagationSpeed`
`SampleRate`	`SweepSlope`
`DechirpInput`	`RangeFFTLength`
`ReferenceRangeCentered`	`ReferenceRange`
`PRFSource`	`PRF`
`DopplerFFTLength`	`DopplerOutput`

Properties	
OperatingFrequency	NormalizeDoppler
SpeedUnits	FrequencyUnits

Properties Applicable to Response Data

These properties are applicable when `IQDataInput` is `false`.

Properties	
Name	Position
ResponseUnits	RangeLabel
DopplerLabel	

Version History

Introduced in R2019a

See Also

`show` | `hide` | `isVisible` | `phased.RangeDopplerResponse` | `phased.AngleDopplerScope` | `phased.RangeAngleScope`

phased.RangeAngleScope

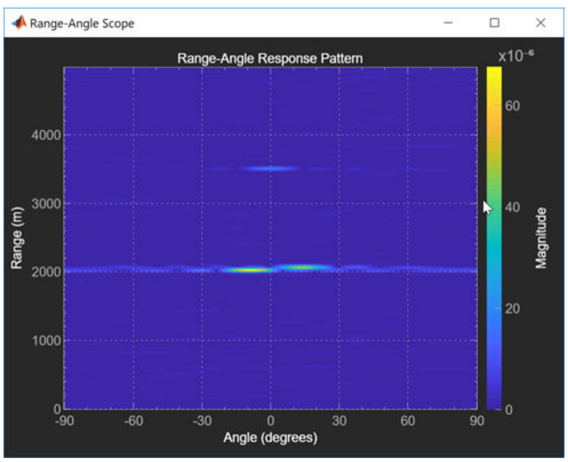
Package: phased

View range-angle response

Description

The phased.RangeAngleScope System object creates a scope for displaying a range-angle response map. The map is a 2-D representation of response intensity as a function of range and angle of arrival. You can input two types of data – in-phase and quadrature (I/Q) data and response data.

- I/Q data – The data consists of fast-time I/Q samples of pulses or sweeps from multiple sensors. The scope computes and displays the response map. To use I/Q data, set the IQDataInput property to true. In this mode, you can set the properties shown in “Properties Applicable to I/Q Data” on page 1-1318.
- Response data – The data consists of the range-angle response itself. The scope displays the range-angle response map. You can obtain range-angle response data from the RangeAngleResponse object. To use response data, set the IQDataInput property to false. In this mode, you can set the properties shown in “Properties Applicable to Response Data” on page 1-1319.



To display a range-angle response map using a scope,

1. Create the phased.RangeAngleScope object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
scope = phased.RangeAngleScope
```

```
scope = phased.RangeAngleScope(Name,Value)
```

Description

`scope = phased.RangeAngleScope` creates a range-angle scope System object for displaying the range-angle response.

`scope = phased.RangeAngleScope(Name,Value)` creates a range-angle `scope` with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose property names in quotes. For example,

```
scope = phased.RangeAngleScope('IQInputData',true,'RangeMethod', ...
        'FFT','SampleRate',1e6)
```

creates a scope object that uses FFT-based range processing to process I/Q data with a sample rate of 1 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

Name — Display caption
`'Range-Angle Scope'` (default) | character vector

Display caption, specified as a character vector. The caption appears in the title bar of the window.

Example: `'Aircraft Range-Angle Response'`

Tunable: Yes

Data Types: `char`

Position — Location and size of intensity scope window
depends on display-resolution (default) | 1-by-4 vector of positive values

Location and size of the intensity scope window, specified as a 1-by-4 vector having the form `[left bottom width height]`.

- `left` and `bottom` specify the location of the bottom-left corner of the window.
- `width` and `height` specify the width and height of the window.

Units are in pixels.

The default value of this property depends on the resolution of your display. By default, the window is positioned in the center of the screen, with a width and height of 800 and 450 pixels, respectively.

Example: `[100 100 500 400]`

Tunable: Yes

Data Types: `double`

IQDataInput — Type of input data
true (default) | false

Type of input data, specified as true or false. When true, the object assumes that the input consists of I/Q sample data and further processing is required in the range and angle domains. When false, the object assumes that the data is response data that has already been processed.

Data Types: logical

ResponseUnits — Response units label
'db' (default) | 'magnitude' | 'power'

Response units, specified as 'db', 'magnitude', or 'power'.

Data Types: char

RangeLabel — Range-axis label
'Range (m)' (default) | character vector

Range-axis label, specified as a character vector.

Example: 'Range (km)'

Tunable: Yes

Dependencies

To enable this property, set the IQDataInput to false.

Data Types: char

AngleLabel — Angle-axis label
'Angle (degrees)' (default) | character vector

Angle-axis label, specified as a character vector.

Example: 'Angle Span (degrees)'

Tunable: Yes

Dependencies

To enable this property, set the IQDataInput to false.

Data Types: char

SensorArray — Sensor array
phased.ULA array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object. See phased.ULA for the default values of a uniform linear array.

Example: phased.URA

Dependencies

To enable this property, set the IQDataInput to true.

RangeMethod — Range processing method
'Matched filter' (default) | 'FFT'

Range-processing method, specified as 'Matched filter' or 'FFT'.

'Matched filter'	The object applies a matched filter to the incoming signal. This approach is commonly used with pulsed signals, where the matched filter is a time-reversed replica of the transmitted signal.
'FFT'	Algorithm performs range processing by applying an FFT to the input signal. This approach is commonly used with FMCW continuous signals and linear FM pulsed signals.

Dependencies

To enable this property, set the IQDataInput property to true.

RangeUnits — Range units
'm' (default) | 'km' | 'mi' | 'nmi'

Range units, specified as:

- 'm' – meters
- 'km' – kilometers
- 'mi' – miles
- 'nmi' – nautical miles

Example: 'mi'

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: char

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value of this property is the speed of light. See physconst. Units are in meters/second.

Example: 3e8

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: double

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: `double`

`SampleRate` — Sample rate
`1e6` (default) | positive scalar

Sample rate, specified as a positive scalar. Units are in Hz.

Example: `10e3`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`SweepSlope` — FM sweep slope
`1e9` (default) | scalar

Slope of the linear FM sweep, specified as a scalar. Units are in Hz/sec.

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'FFT'`.

Data Types: `double`

`DechirpInput` — Dechirp input signal
`false` (default) | `true`

Set this property to `true` to dechirp the input signal before performing range processing. `false` indicates that the input signal is already dechirped and no dechirp operation is necessary.

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'FFT'`.

Data Types: `logical`

`RangeFFTLength` — FFT length used in range processing
`1024` (default) | positive integer

FFT length used for range processing, specified as a positive integer.

Example: `128`

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'FFT'`.

Data Types: `double`

`ReferenceRangeCentered` — Set reference range at center of range span
`true` (default) | `false`

Set this property to `true` to set the reference range to the center of the range span. Set this property to `false` to set the reference range to the beginning of the range span.

Dependencies

To enable this property, set the IQDataInput property to true and the RangeMethod property to 'FFT'.

Data Types: logical

ReferenceRange — Reference range
0.0 (default) | nonnegative scalar

Reference range of the range span, specified as a nonnegative scalar.

- If you set the RangeMethod property to 'Matched filter', the reference range marks the start of the range span.
- If you set the RangeMethod property to 'FFT', the position of the reference range depends on the ReferenceRangeCentered property.
  - If you set the ReferenceRangeCentered property to true, the reference range marks the center of the range span.
  - If you set the ReferenceRangeCentered property to false, the reference range marks the start of the range span.

  Units are in meters.

Example: 1000.0

Tunable: Yes

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: double

ElevationAngle — Elevation angle of response
0 (default) | scalar

Elevation angle at which to calculate the response, specified as a scalar. The elevation angle must lie in the interval from –90° to 90°, inclusive. Units are in degrees.

Example: 45.0

Dependencies

To enable this property, set the IQDataInput property to true.

Data Types: double

AngleSpan — Azimuth angle span of response
[-90 90] (default) | real-valued 1-by-2 vector

Azimuth angle span at which to calculate response, specified as a real-valued 1-by-2 row vector. The object calculates the range-angle response within the angle range, [min_angle max_angle]. Angles must lie in the interval from –90° to 90°, inclusive. Units are in degrees.

Example: [-45 45]

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`NumAngleSamples` — Number of samples in azimuth angle span
`256` (default) | positive integer greater than two

Number of samples in the azimuth angle span at which to calculate the range-angle response, specified as a positive integer greater than two.

Example: `256`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

Usage

Syntax

```
scope(X,Range,Ang)
```

```
scope(X)
scope(X,XREF)
scope(X,COEFF)
```

Description

`scope(X,Range,Ang)` displays a range-angle response map, `X`, at the ranges, `Range`, and angles, `Ang`. This syntax applies when you set the `IQDataInput` to `false`.

`scope(X)` computes and displays the range-angle response map for the dechirped signal `X`. This syntax applies when you set the `IQDataInput` property to `true`, the `RangeMethod` property to `'FFT'`, and the `DechirpInput` property to `false`. This syntax is most commonly used with FMCW signals.

`scope(X,XREF)` also specifies a reference signal to use for dechirping the input signal, `X`. This syntax applies when you set the `IQDataInput` property to `true`, the `RangeMethod` property to `'FFT'`, and the `DechirpInput` property to `true`. This syntax is most commonly used with FMCW signals. `XREF` is generally the transmitted signal.

`scope(X,COEFF)` also specifies matched filter coefficients, `COEFF`. This syntax applies when you set the `IQDataInput` property to `true` and the `RangeMethod` property to `'Matched Filter'`. This syntax is most commonly used with pulsed signals.

Input Arguments

`X` — Input data
complex-valued *K*-by-*L* matrix

Input data, specified as a complex-valued *K*-by-*L* matrix. The interpretation of the data depends on the value of the `IQDataInput` property.

- When IQDataInput is true, the input consists of received fast-time data samples for each PRI pulse or FMCW sweep and for each array or subarray element. *K* denotes the number of fast-time (range) samples. *L* is the number of elements. If SensorArray contains subarrays, *L* is the number of subarrays. The scope computes and displays the range-angle response.
  - When RangeMethod is set to 'FFT' and DechirpInput is false, X has previously been dechirped.
  - When RangeMethod is set to 'FFT' and DechirpInput is true, X has not been previously dechirped. Use the syntax that includes XREF as input data.
  - When RangeMethod is set to 'MatchedFilter', X has not been match filtered. Use the syntax that includes COEFF as input data.
- When IQDataInput is false, the input already consists of response data in the range-angle domain, such as the data produced, for example, by RangeAngleResponse. Each row of the response map corresponds to an element of the Range vector. *K* is the number of range samples. Each column of the response map corresponds to an element of the Ang vector. *L* is the number of angles. The scope serves only as a display of the range-angle response.

Range — Range grid values of range-angle response map
real-valued *K*-by-1 column vector

Range grid values of range-angle response map, specified as a real-valued *K*-by-1 column vector. Range denotes the range values at which the response has been computed. Elements of Range correspond to the rows of X.

Dependencies

To enable this argument, set the IQInputData property to false.

Data Types: double

Ang — Angle grid values of range-angle response map
real-valued *L*-by-1 column vector

Angle grid values of response map, specified as a real-valued *K*-by-1 column vector. Ang denotes the angle values at which the response has been computed. Elements of Ang correspond to the columns of X.

Dependencies

To enable this argument, set the IQInputData property to false.

Data Types: double

XREF — Reference signal
complex-valued K-by-1 column vector

Reference signal used to dechirp X. XREF must be a column vector with the same number of rows as X.

Dependencies

To enable this argument, set the IQDataInput property to true, the RangeMethod property to 'FFT' and the DechirpInput property to false

Data Types: double
Complex Number Support: Yes

COEFF — Matched filter coefficients
complex-valued column vector

Matched filter coefficients, specified as a complex-valued column vector.

Dependencies

To enable this argument, set the `IQDataInput` property to `true` and the `RangeMethod` property to `'Matched Filter'`.

Data Types: `double`
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Scope Objects

show	Turn on visibility of scopes
hide	Turn off visibility of scope
isVisible	Visibility of scopes

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

View Target Response Using Range-Angle Scope

Calculate and visualize the range-angle response from a pulsed radar transmitting a rectangular waveform using a matched filter. One target is approximately 2000 m away and is stationary relative to the radar. The second target is approximately 3500 m away and is also stationary relative to the radar. The third is approximately 2000 m away and is moving away from the radar at approximately 100 m/s. The signals arrive at an 8-element uniform linear array.

Load the data to obtain signals and parameters.

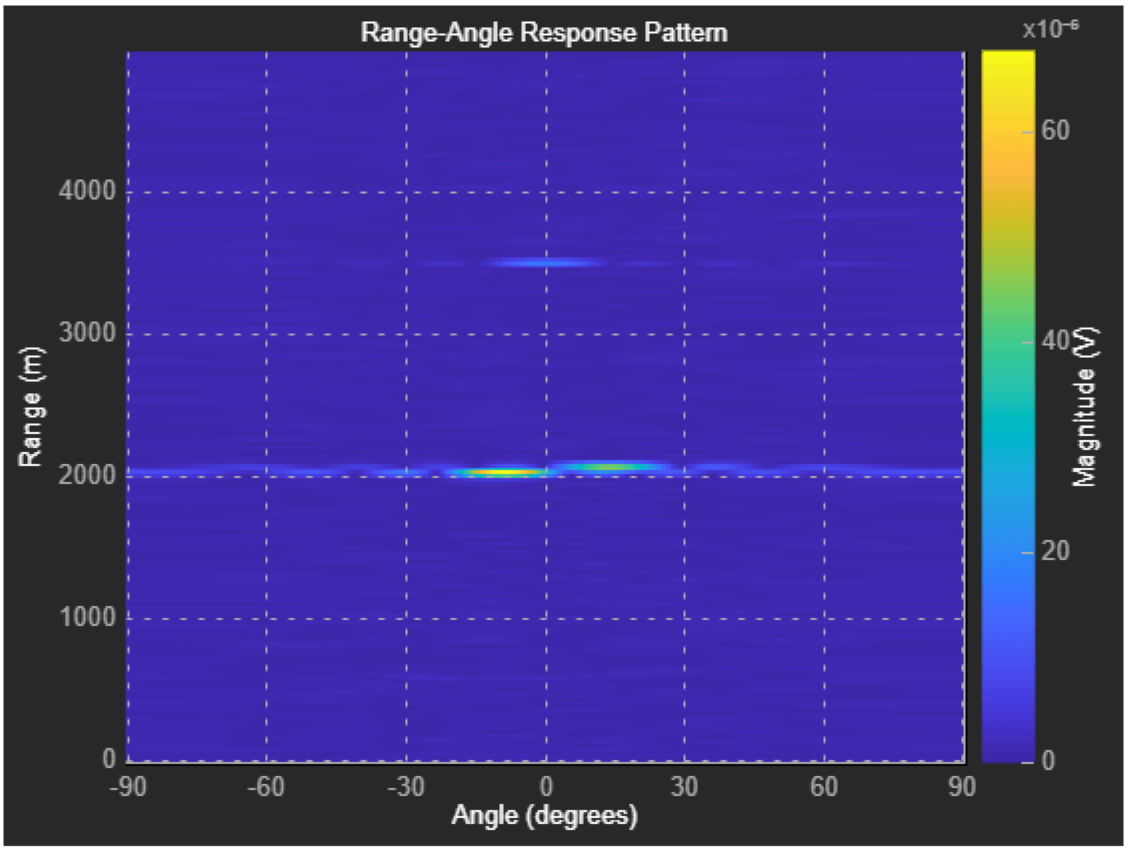
```
load('RangeAngleResponseExampleData','rectdata');
fs = rectdata.fs;
c = rectdata.propspeed;
fc = rectdata.fc;
rxdata = rectdata.rxdata;
mfcoeffs = rectdata.mfcoeffs;
noisepower = rectdata.noisepower;
array = rectdata.antennaarray;
```

Create a range-angle scope for processing.

```
scope = phased.RangeAngleScope( ...
    'IQDataInput',true,'RangeMethod','Matched filter', ...
    'Name','Range-Angle Scope','ResponseUnits','magnitude', ...
    'Position',[560 375 560 420],'RangeUnits','m', ...
    'SensorArray',array,'OperatingFrequency',fc, ...
    'SampleRate',fs,'PropagationSpeed',c);
```

Call the scope to display the response map.

```
scope(rxdata,mfcoeffs)
```



More About

Properties Applicable to I/Q Data

These properties are applicable when `IQDataInput` is `true`.

Properties	
Name	Position
ResponseUnits	SensorArray
RangeMethod	PropagationSpeed
OperatingFrequency	RangeUnits

Properties	
SampleRate	SweepSlope
DechirpInput	RangeFFTLength
ReferenceRangeCentered	ReferenceRange
ElevationAngle	AngleSpan
NumAngleSamples	

Properties Applicable to Response Data

These properties are applicable when `IQDataInput` is `false`.

Properties	
Name	Position
ResponseUnits	RangeLabel
AngleLabel	

Version History

Introduced in R2019a

See Also

`show` | `hide` | `isVisible` | `phased.RangeDopplerScope` | `phased.AngleDopplerScope` | `RangeAngleResponse`

phased.AngleDopplerScope

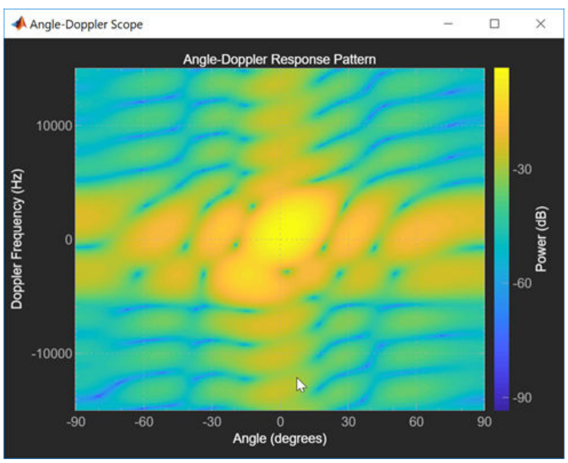
Package: phased

Angle-Doppler scope

Description

The phased.AngleDopplerScope System object creates a scope for displaying an angle-Doppler response map. The map is a 2-D representation of response intensity as a function of angle and Doppler shift. You can input two types of data - in-phase and quadrature (I/Q) data and response data.

- I/Q data – The data consists of I/Q samples at the same range from multiple sensors over all pulses or sweeps. The scope computes and displays the response map. To use I/Q data, set the IQDataInput property to true. In this mode, you can set the properties listed in “Properties Applicable to I/Q Data” on page 1-1328.
- Response data – The data consists of the angle-Doppler response itself. The scope only displays the angle-Doppler response map. You can obtain angle-Doppler response data from the phased.AngleDopplerResponse object. To display response data, set the IQDataInput property to false. In this mode, you can set the properties listed in “Properties Applicable to Response Data” on page 1-1328.



To display an angle-Doppler response map using a scope,

1. Create the phased.AngleDopplerScope object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
scope = phased.AngleDopplerScope
```

```
scope = phased.phased.AngleDopplerScope(Name,Value)
```

Description

`scope = phased.AngleDopplerScope` creates an angle-Doppler `scope` System object for displaying the angle-Doppler response map.

`scope = phased.phased.AngleDopplerScope(Name,Value)` creates an angle-Doppler `scope` with each specified property set to the specified value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose property names in quotes. For example,

```
scope = phased.AngleDopplerScope('IQInputData',true, ...
      'NumAngleSamples',128,'NumDopplerSamples',64)
```

creates a scope object that computes and displays the angle-Doppler response at 128 angle values and 64 Doppler values from I/Q data input.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

Name — Display caption

`'Angle-Doppler Scope'` (default) | character vector

Display caption, specified as a character vector. The caption appears in the title bar of the window.

Example: `'Aircraft Angle-Doppler Response'`

Tunable: Yes

Data Types: `char`

Position — Location and size of intensity scope window

depends on display-resolution (default) | 1-by-4 vector of positive values

Location and size of the intensity scope window, specified as a 1-by-4 vector having the form `[left bottom width height]`.

- `left` and `bottom` specify the location of the bottom-left corner of the window.
- `width` and `height` specify the width and height of the window.

Units are in pixels.

The default value of this property depends on the resolution of your display. By default, the window is positioned in the center of the screen, with a width and height of 800 and 450 pixels, respectively.

Example: `[100 100 500 400]`

Tunable: Yes

Data Types: double

IQDataInput — Type of input data
true (default) | false

Type of input data, specified as true or false. When true, the object assumes that the input consists of I/Q sample data and further processing is required in the range, angle, or Doppler domains. When false, the object assumes that the data is response data that has already been processed.

Data Types: logical

ResponseUnits — Response units label
'db' (default) | 'magnitude' | 'power'

Response units, specified as 'db', 'magnitude', or 'power'.

Data Types: char

AngleLabel — Angle-axis label
'Angle (degrees)' (default) | character vector

Angle-axis label, specified as a character vector.

Example: 'Angle Span (degrees)'

Tunable: Yes

Dependencies

To enable this property, set the IQDataInput to false.

Data Types: char

DopplerLabel — Doppler-axis label
'Doppler Frequency (Hz)' (default) | character vector

Doppler-axis label, specified as a character vector.

Example: 'Doppler Frequency (kHz)'

Tunable: Yes

Dependencies

To enable this property, set the IQDataInput to false.

Data Types: char

SensorArray — Sensor array
phased.ULA array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object. See phased.ULA for the default values of a uniform linear array.

Example: phased.URA

Dependencies

To enable this property, set the `IQDataInput` to `true`.

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value of this property is the speed of light. See `physconst`. Units are in meters/second.

Example: `3e8`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

PRF — Pulse repetition frequency of input signal
`1` (default) | positive scalar

Pulse repetition frequency of input signal, specified as a positive scalar. Units are in Hz.

Example: `1.4e3`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`ElevationAngle` — Elevation angle of response
`0` (default) | scalar

Elevation angle at which to calculate the response, specified as a scalar. The elevation angle must lie in the interval from –90° to 90°, inclusive. Units are in degrees.

Example: `45.0`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`NumAngleSamples` — Number of bins in angle span
`256` (default) | positive integer greater than two

Number of bins in the angle span at which to calculate the response, specified as a positive integer greater than two.

Example: 256

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`NumDopplerSamples` — Number of Doppler bins
`256` (default) | positive integer greater than two

Number of bins in the Doppler domain used to calculate angle-Doppler response, specified as a positive integer greater than two.

Example: `512`

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `double`

`NormalizeDoppler` — Normalize Doppler
`false` (default) | `true`

Set this property to `true` to plot the angle-Doppler response at the normalized Doppler frequency. Set this property to `false` to plot the angle-Doppler response without normalizing the Doppler frequency.

Dependencies

To enable this property, set the `IQDataInput` property to `true`.

Data Types: `logical`

`FrequencyUnits` — Doppler frequency units
`'Hz'` (default) | `'kHz'` | `'MHz'`

Doppler frequency units, specified as `'Hz'`, `'kHz'`, or `'MHz'`.

Example: `'MHz'`

Dependencies

To enable this property, set the `IQDataInput` property to `true` and the `NormalizedDoppler` property to `false`.

Data Types: `char`

Usage

Syntax

`scope(X,Ang,Dop)`

`scope(X)`

Description

scope(X,Ang,Dop) displays an angle-Doppler response map for the response data, scope, for direction azimuth angles, Ang, and Doppler shifts, Dop. This syntax applies when you set the IQDataInput to false.

scope(X) computes and displays the angle-Doppler response map of the I/Q data X. This syntax applies when you set the IQDataInput property to true.

Input Arguments

X — Input data

real-valued *P*-by-*Q* matrix | complex-valued *P*-by-*Q* matrix | complex-valued *L*-by-1 column vector

Input data, specified as a real-valued *P*-by-*Q* or complex-valued *P*-by-*Q* matrix. The processing of the data depends on the value of the IQDataInput property.

- When IQDataInput is true, x consists of I/Q samples at fixed range of pulses or sweeps from multiple elements or subarrays. *P* is the number of array elements. If SensorArray contains subarrays, *P* is the number of subarrays. *Q* is the number of pulses. The scope computes and displays the angle-Doppler response.

  When x is a column vector, *L* must be equal to an integer multiple of *P*.
- When IQDataInput is false, x consists of real-valued angle-Doppler response data such as the data produced by phased.AngleDopplerResponse. *P* is the number of Doppler samples and *Q* is the number of angle samples. Each row represents a Doppler value corresponding to an element of Dop. Each column represents an angle value corresponding to an element of the Ang vector. The scope serves only as a display of the angle-Doppler response.

Ang — Azimuth angle grid values of response map

real-valued *Q*-by-1 column vector

Azimuth angle grid values of response map, specified as a real-valued *Q*-by-1 column vector. Ang contains the angle values corresponding to the columns of X.

Dependencies

To enable this argument, set the IQInputData property to false.

Data Types: double

Dop — Doppler grid values of response map

real-valued *P*-by-1 column vector

Doppler grid values of response map, specified as a real-valued *P*-by-1 column vector. Dop contains the Doppler values corresponding to the rows of X.

Dependencies

To enable this argument, set the IQInputData property to false.

Data Types: double

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to Scope Objects

show	Turn on visibility of scopes
hide	Turn off visibility of scope
isVisible	Visibility of scopes

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

View Target Response Using Angle-Doppler Scope

Calculate and visualize the angle-Doppler response at a single range cell of a collected data cube.

Load the I/Q data and analyze the 43th range cell.

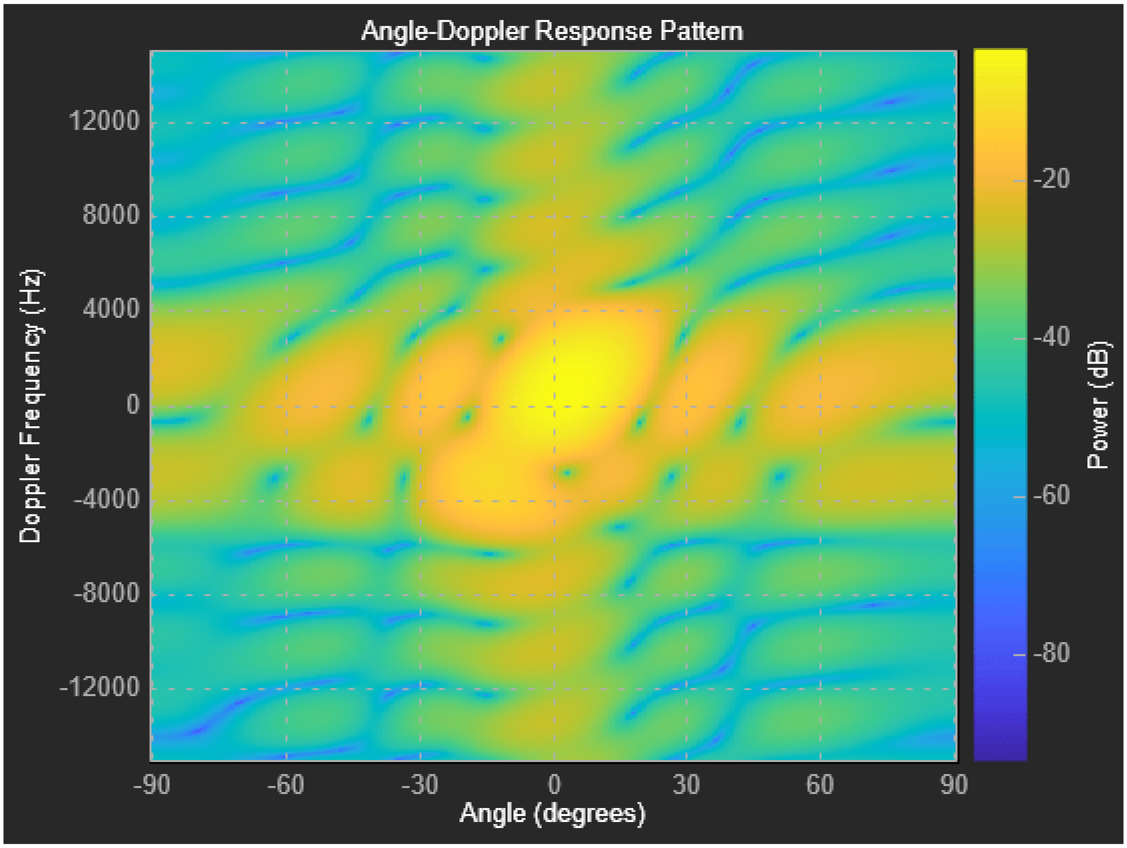
```
load STAPExampleData;
x = shiftdim(STAPEx_ReceivePulse(43,:,:));
```

Create a scope object that processes I/Q data.

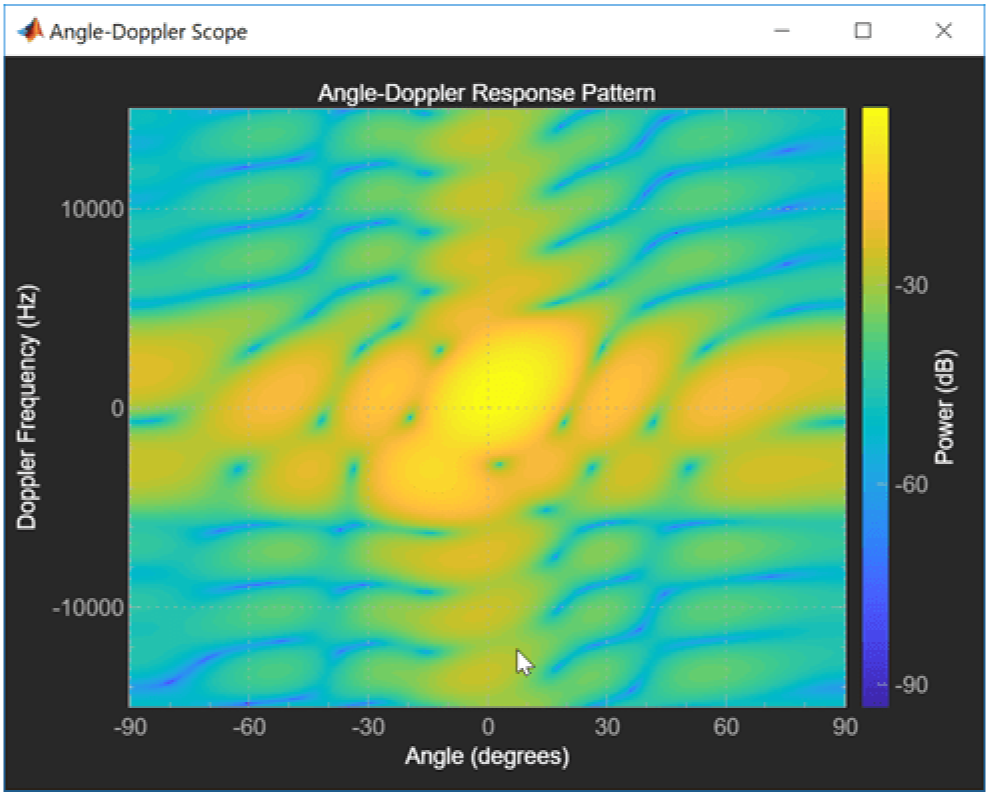
```
scope = phased.AngleDopplerScope( ...
    'IQDataInput', true, ...
    'Name','Angle-Doppler Scope', ...
    'Position',[560 375 560 420], ...
    'NormalizeDoppler',false, ...
    'ResponseUnits','db', ...
    'SensorArray',STAPEx_HArray, ...
    'OperatingFrequency',STAPEx_OperatingFrequency, ...
    'PropagationSpeed',STAPEx_PropagationSpeed, ...
    'PRF',STAPEx_PRF,'NumDopplerSamples',512);
```

Compute and visualize the angle-Doppler response.

```
scope(x)
```



Angle-Doppler Response Pattern
Doppler Frequency (Hz)
12000
8000
4000
0
-4000
-8000
-12000
-90
-60
-30
0
30
60
90
Angle (degrees)
-20
-40
-60
-80
Power (dB)



More About

Properties Applicable to I/Q Data

These properties are applicable when `IQDataInput` is `true`.

Properties	
`Name`	`Position`
`ResponseUnits`	`SensorArray`
`PropagationSpeed`	`OperatingFrequency`
`NumAngleSamples`	`NumDopplerSamples`
`PRF`	`ElevationAngle`
`NormalizeDoppler`	`FrequencyUnits`

Properties Applicable to Response Data

These properties are applicable when `IQDataInput` is `false`.

Properties	
`Name`	`Position`
`ResponseUnits`	`DopplerLabel`

Properties	
AngleLabel	

Version History

Introduced in R2019a

See Also

show | hide | isVisible | phased.AngleDopplerResponse | phased.RangeAngleScope | phased.RangeDopplerScope

hide

Package: `phased`

Turn off visibility of scope

Syntax

`hide(scope)`

Description

`hide(scope)` hides the display window of the `scope` System object.

Input Arguments

`scope` — Scope system object
scope System object

Scope, specified as a scope System object such as `RangeDopplerScope`.

Version History

Introduced in R2019a

See Also

`show` | `isVisible` | `phased.RangeAngleScope` | `phased.RangeDopplerScope` | `phased.AngleDopplerScope` | `phased.RTIScope` | `phased.DTIScope`

isVisible

Package: `phased`

Visibility of scopes

Syntax

`vis = isVisible(scope)`

Description

`vis = isVisible(scope)` returns the visibility of the `scope` System object.

Input Arguments

`scope` — Scope system object
scope System object

Scope, specified as a scope System object such as `RangeDopplerScope`.

Output Arguments

`vis` — Visibility of scope
`true` | `false`

Visibility of scope, returned as `true` or `false`. When `true`, `scope` is visible. When `false`, `scope` is hidden.

Data Types: `logical`

Version History

Introduced in R2019a

See Also

`show` | `hide` | `phased.RangeAngleScope` | `phased.RangeDopplerScope` | `phased.AngleDopplerScope` | `phased.RTIScope` | `phased.DTIScope`

show

Package: `phased`

Turn on visibility of scopes

Syntax

`show(scope)`

Description

`show(scope)` shows the display window of the `scope` System object.

Input Arguments

`scope` — Scope system object
scope System object

Scope, specified as a scope System object such as `RangeDopplerScope`.

Version History

Introduced in R2019a

See Also

`hide` | `isVisible` | `phased.RangeAngleScope` | `phased.RangeDopplerScope` | `phased.AngleDopplerScope` | `phased.RTIScope` | `phased.DTIScope`

phased.RangeDopplerResponse

Package: `phased`

Range-Doppler response

Description

The `phased.RangeDopplerResponse` System object calculates the filtered response to fast-time and slow-time data. or equivalently, range data, using either a matched filter or an FFT.

The input to the Doppler response object is a data cube. The organization of the data cube follows the Phased Array System Toolbox convention. The first dimension of the cube represents the fast-time samples or ranges of the received signals. The second dimension represents multiple channels such as sensors or beams. The third dimension, slow time, represent pulses. If the data contains only one channel or pulse, the data cube can contain fewer than three dimensions. Range processing operates along the first dimension of the cube. Doppler processing operates along the last dimension.

The output of the object is also a data cube with the same number of dimensions as the input. The first dimension contains range-processed data but its length can differ from the first dimension of the input. The last dimension contains Doppler processed data. Its length can differ from the last dimension of the input.

To compute the range-Doppler response:

1 Define and set up your `phased.RangeDopplerResponse` System object. See "Construction" on page 1-1333.
2 Call `step` to compute the range-Doppler response of the input signal according to the properties of `phased.RangeDopplerResponse`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.RangeDopplerResponse` creates a range-Doppler response System object, `H`. The object calculates the range-Doppler response of the input data.

`H = phased.RangeDopplerResponse(Name,Value)` creates a range-Doppler response object, `H`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name on page 1-1334, and `Value` is the corresponding value. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,…,NameN,ValueN`.

Properties

`RangeMethod`

Range processing method

Specify the method of range processing as `'Matched filter'` or `'FFT'`.

`'Matched filter'`	Algorithm applies a matched filter to the incoming signal. This approach is common with pulsed signals, where the matched filter is the time reverse of the transmitted signal.
`'FFT'`	Algorithm performs range processing by applying an FFT to the input signal. This approach is commonly used with FMCW and linear FM pulsed signals.

Default: `'Matched filter'`

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

`SampleRate`

Sample rate

Specify the sample rate, in hertz, as a positive scalar. This property can be specified as single or double precision. The default value corresponds to 1 MHz.

Default: `1e6`

`SweepSlope`

FM sweep slope

Specify the slope of the linear FM sweeping, in hertz per second, as a scalar. The `x` data you provide to `step` or `plotResponse` must correspond to sweeps having this slope. This property can be specified as single or double precision.

To enable this property, set the `RangeMethod` property to `'FFT'`.

Default: `1e9`

`DechirpInput`

Option to dechirp input signal

Set this property to `true` to have the range-Doppler response object dechirp the input signal. Set this property to `false` to indicate that the input signal is already dechirped and no dechirp operation is necessary.

To enable this property, set the `RangeMethod` property to `'FFT'`.

Default: `false`

DecimationFactor

Decimation factor for dechirped signal

Specify the decimation factor for the dechirped signal as a positive integer. When processing FMCW signals, you can often decimate the dechirped signal to reduce the requirements on the analog-to-digital converter.

To enable this property, set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `true`. This property can be specified as single or double precision. The default value indicates no decimation.

Default: `1`

RangeFFTLengthSource

Source of FFT length used in range processing

Specify how the object determines the FFT length used in range processing. Values of this property are:

`'Auto'`	The FFT length equals the number of rows of the input signal.
`'Property'`	The `RangeFFTLength` property of this object specifies the FFT length.

To enable this property, set the `RangeMethod` property to `'FFT'`.

Default: `'Auto'`

RangeFFTLength

FFT length in range processing

Specify the FFT length in the range domain as a positive integer. This property can be specified as single or double precision.

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeFFTLengthSource` property to `'Property'`.

Default: `1024`

RangeWindow

Window for range weighting

Specify the window used for range processing using one of `'None'`, `'Hamming'`, `'Chebyshev'`, `'Hann'`, `'Kaiser'`, `'Taylor'`, or `'Custom'`. If you set this property to `'Taylor'`, the generated Taylor window has four nearly constant sidelobes adjacent to the mainlobe.

To enable this property, set the `RangeMethod` property to `'FFT'`.

Default: `'None'`

RangeSidelobeAttenuation

Sidelobe attenuation level for range processing

Specify the sidelobe attenuation level of a Kaiser, Chebyshev, or Taylor window in range processing as a positive scalar, in decibels. This property can be specified as single or double precision.

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeWindow` property to `'Kaiser'`, `'Chebyshev'`, or `'Taylor'`.

Default: `30`

CustomRangeWindow

User-defined window for range processing

Specify the user-defined window for range processing using a function handle or a cell array.

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeWindow` property to `'Custom'`.

If `CustomRangeWindow` is a function handle, the specified function takes the window length as the input and generates appropriate window coefficients.

If `CustomRangeWindow` is a cell array, then the first cell must be a function handle. The specified function takes the window length as the first input argument, with other additional input arguments, if necessary. The function then generates appropriate window coefficients. The remaining entries in the cell array are the additional input arguments to the function, if any.

Default: `@hamming`

ReferenceRangeCentered

Set reference range at center of range grid, specified as `true` or `false`. Setting this property to `true` enables you to set the reference range at the center of the range grid. Setting this property to `false` sets the reference range to the beginning of the range grid.

Dependencies

To enable this property, set the `RangeMethod` to `'FFT'`.

Default: `true`

ReferenceRange

Reference range of the range grid, specified as a nonnegative scalar.

- If you set the `RangeMethod` property to `'Matched filter'`, the reference range is set to the start of the range grid.
- If you set the `RangeMethod` property to `'FFT'`, the reference range is determined by the `ReferenceRangeCentered` property.
  - When you set the `ReferenceRangeCentered` property to `true`, the reference range is set to the center of the range grid.
  - When you set the `ReferenceRangeCentered` property to `false`, the reference range is set to the start of the range grid.

This property can be specified as single or double precision. Units are in meters.

This property is tunable.

Example: 1000.0

Default: `0.0`

PRFSource

Source of pulse repetition frequency

Source of pulse repetition frequency, specified as

- `'Auto'` — You assume that the pulse repetition frequency (PRF) is the inverse of the duration of the input signal to the `step` method. Then the PRF equals the sample rate of the signal divided by the number of rows in the input signal.
- `'Property'`— specify the pulse repetition frequency using the `PRF` property.
- `'Input port'`— specify the PRF using an input argument of the `step` method.

Use the `'Property'` or `'Input port'` option when the pulse repetition frequency cannot be determined by the signal duration, as is the case with range-gated data.

Default: `'Auto'`

PRF

Pulse repetition frequency of input signal

Pulse repetition frequency of the input signal, specified as a positive scalar. `PRF` must be less than or equal to the sample rate divided by the number of rows of the input signal to the `step` method. When the signal length is variable, use the maximum possible number of rows of the input signal instead. This property can be specified as single or double precision.

To enable this property, set the `PRFSource` property to `'Property'`.

Default: `10e3`

DopplerFFTLengthSource

Source of FFT length in Doppler processing

Specify how the object determines the FFT length in Doppler processing. Values of this property are:

`'Auto'`	The FFT length is equal to the number of rows of the input signal.
`'Property'`	The `DopplerFFTLength` property of this object specifies the FFT length.

To enable this property, set the `RangeMethod` property to `'FFT'`.

Default: `'Auto'`

DopplerFFTLength

FFT length for Doppler processing

FFT length for Doppler processing, specified as a positive integer. This property can be specified as single or double precision.

To enable this property, set the `RangeMethod` property to `'FFT'` and the `DopplerFFTLengthSource` property to `'Property'`.

Default: `1024`

`DopplerWindow`

Window for Doppler weighting

Specify the window used for Doppler processing using one of `'None'`, `'Hamming'`, `'Chebyshev'`, `'Hann'`, `'Kaiser'`, `'Taylor'`, or `'Custom'`. If you set this property to `'Taylor'`, the generated Taylor window has four nearly constant sidelobes adjacent to the mainlobe.

To enable this property, set the `RangeMethod` property to `'FFT'`.

Default: `'None'`

`DopplerSidelobeAttenuation`

Sidelobe attenuation level for Doppler processing

Specify the sidelobe attenuation level of a Kaiser, Chebyshev, or Taylor window in Doppler processing as a positive scalar, in decibels. This property can be specified as single or double precision.

To enable this property, set the `RangeMethod` property to `'FFT'` and the `DopplerWindow` property to `'Kaiser'`, `'Chebyshev'`, or `'Taylor'`.

Default: `30`

`CustomDopplerWindow`

User-defined window for Doppler processing

Specify the user-defined window for Doppler processing using a function handle or a cell array..

If `CustomDopplerWindow` is a function handle, the specified function takes the window length as the input and generates appropriate window coefficients.

If `CustomDopplerWindow` is a cell array, then the first cell must be a function handle. The specified function takes the window length as the first input argument, with other additional input arguments, if necessary. The function then generates appropriate window coefficients. The remaining entries in the cell array are the additional input arguments to the function, if any.

To enable this property, set the `RangeMethod` property to `'FFT'` and the `DopplerWindow` property to `'Custom'`

Default: `@hamming`

`DopplerOutput`

Doppler domain output

Specify the Doppler domain output as `'Frequency'` or `'Speed'`. The Doppler domain output is the `DOP_GRID` argument of `step`.

`'Frequency'`	`DOP_GRID` is the Doppler shift, in hertz.
`'Speed'`	`DOP_GRID` is the radial speed corresponding to the Doppler shift, in meters per second.

Default: `'Frequency'`

OperatingFrequency

Signal carrier frequency

Specify the carrier frequency, in hertz, as a scalar. The default value of this property corresponds to 300 MHz. This property can be specified as single or double precision.

To enable this property, set the `DopplerOutput` property to `'Speed'`

Default: `3e8`

MaximumNumInputSamplesSource

Source of maximum number of samples

The source of the maximum number of samples of the input signal, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the object automatically allocates enough memory to buffer the first input signal. When you set this property to `'Property'`, specify the maximum number of samples in the input signal using the `MaximumNumInputSamples` property. Any input signal longer than that value is truncated.

Default: `'Auto'`

MaximumNumInputSamples

Maximum number of input signal samples

Maximum number of samples in the input signal, specified as a positive integer. This property limits the size of the input signal. Any input signal longer than this value is truncated. The input signal is the first argument to the `step` method. The number of samples is the number of rows in the input. This property can be specified as single or double precision.

To enable this property, set the `RangeMethod` property to `'Matched filter'` and set the `MaximumNumInputSamplesSource` property to `'Property'`.

Default: `100`

Methods

plotResponse	Plot range-Doppler response
step	Calculate range-Doppler response

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Range-Doppler Response Using Matched Filter

Compute the range-doppler response of a pulsed radar signal using a matched filter.

Load data for a pulsed radar signal. The signal includes three target returns. Two targets are approximately 2000 m away, while the third is approximately 3500 m away. In addition, two of the targets are stationary relative to the radar. The third is moving away from the radar at about 100 m/s.

```
load RangeDopplerExampleData;
```

Create a range-Doppler response object.

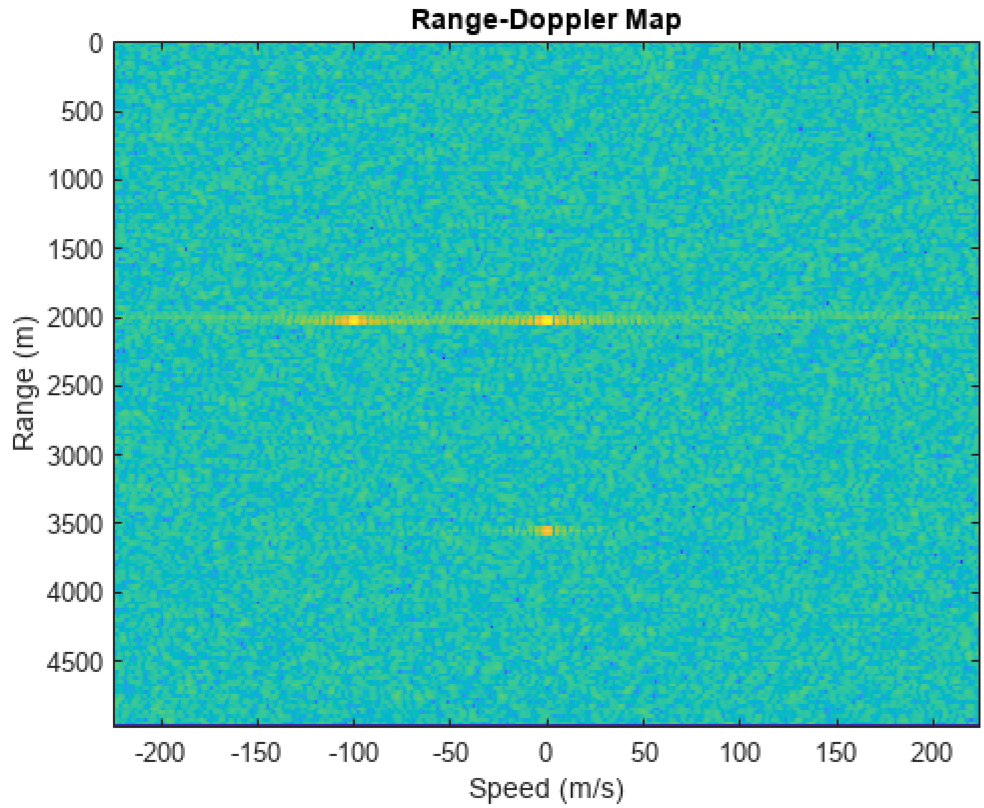
```
response = phased.RangeDopplerResponse('DopplerFFTLengthSource','Property', ...
   'DopplerFFTLength',RangeDopplerEx_MF_NFFTDOP, ...
   'SampleRate',RangeDopplerEx_MF_Fs,'DopplerOutput','Speed', ...
   'OperatingFrequency',RangeDopplerEx_MF_Fc);
```

Calculate the range-Doppler response.

```
[resp,rng_grid,dop_grid] = response(RangeDopplerEx_MF_X, ...
    RangeDopplerEx_MF_Coeff);
```

Plot the range-Doppler response.

```
imagesc(dop_grid,rng_grid,mag2db(abs(resp)));
xlabel('Speed (m/s)');
ylabel('Range (m)');
title('Range-Doppler Map');
```



Range-Doppler Response of FMCW Signal

Compute the range-Doppler response of an FMCW signal using an FFT.

Load data for an FMCW signal that has not been dechirped. The signal contains the return from a target about 2200 m away. The signal has a normalized Doppler frequency of approximately -0.36 relative to the radar.

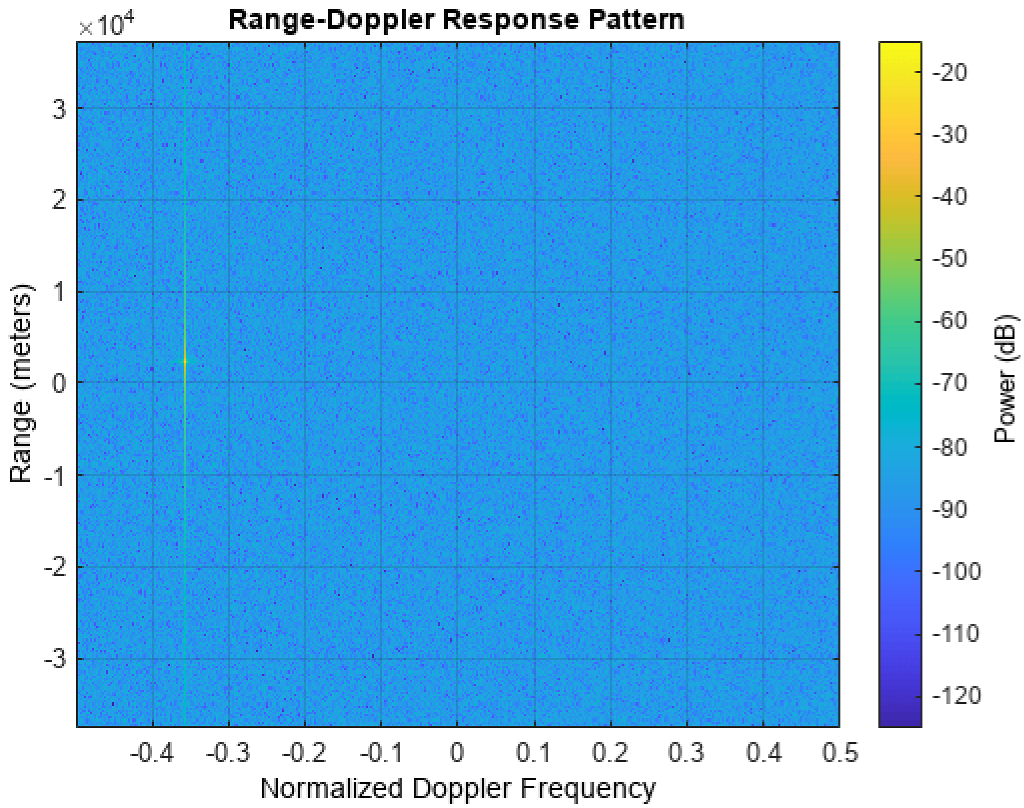
```
load RangeDopplerExampleData;
```

Create a range-Doppler response object.

```
hrdresp = phased.RangeDopplerResponse(...
    'RangeMethod','FFT',...
    'PropagationSpeed',RangeDopplerEx_Dechirp_PropSpeed,...
    'SampleRate',RangeDopplerEx_Dechirp_Fs,...
    'DechirpInput',true,...
    'SweepSlope',RangeDopplerEx_Dechirp_SweepSlope);
```

Plot the range-Doppler response.

```
plotResponse(hrdresp,...
    RangeDopplerEx_Dechirp_X,RangeDopplerEx_Dechirp_Xref,...
    'Unit','db','NormalizeDoppler',true)
```



Estimate Doppler and Range with Specified PRF

Estimate the Doppler and range responses for three targets. Two targets are approximately 2000 m away, while the third is approximately 3500 m away. In addition, two of the targets are stationary relative to the radar. The third is moving away from the radar at about 100 m/s. Specify the pulse repetition frequency.

Load data for a pulsed radar signal.

```
load RangeDopplerExampleData;
```

Create a range-Doppler response object. Set the PRF to 25 kHz.

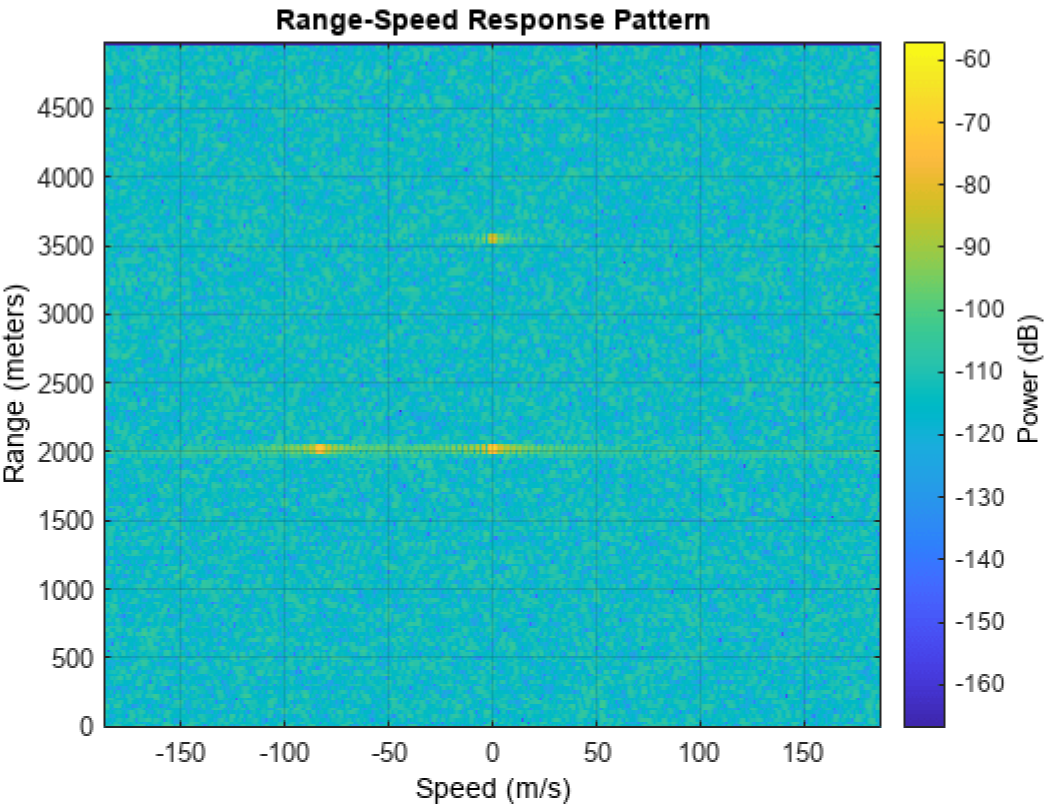
```
response = phased.RangeDopplerResponse('DopplerFFTLengthSource','Property', ...
   'DopplerFFTLength',RangeDopplerEx_MF_NFFTDOP,'SampleRate', ...
   RangeDopplerEx_MF_Fs,'DopplerOutput','Speed','OperatingFrequency', ...
   RangeDopplerEx_MF_Fc,'PRFSource','Property','PRF',25.0e3);
```

Calculate the range-Doppler response.

```
[resp,rng_grid,dop_grid] = response(RangeDopplerEx_MF_X, ...
    RangeDopplerEx_MF_Coeff);
```

Plot the range-Doppler response.

```
plotResponse(response,RangeDopplerEx_MF_X,RangeDopplerEx_MF_Coeff,'Unit','db')
```



Algorithms

Response Algorithm

The `phased.RangeDopplerResponse` object generates a response as follows:

1. Processes the input signal in the fast-time dimension using either a matched filter or dechirp/FFT operation.
2. Processes the input signal in the pulse dimension using an FFT.

The decimation algorithm uses a 30th order FIR filter generated by `fir1(30,1/R)`, where `R` is the value of the `DecimationFactor` property.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2012b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `CustomRangeWindow` and `CustomDopplerWindow` properties are not supported.
- The `plotResponse` method is not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

Functions

`bw2range` | `fir1` | `chebwin` | `dechirp` | `hann` | `hamming` | `kaiser` | `taylorwin` | `rangeangle`

Objects

`phased.RangeAngleResponse` | `phased.RangeResponse` | `phased.AngleDopplerResponse` | `phased.MatchedFilter` | `phased.DopplerEstimator` | `phased.RangeEstimator` | `phased.CFARDetector` | `phased.CFARDetector2D`

Topics

“Automotive Adaptive Cruise Control Using FMCW Technology” (Radar Toolbox)
“Radar Data Cube Concept”

plotResponse

System object: `phased.RangeDopplerResponse`
Package: `phased`

Plot range-Doppler response

Syntax

```
plotResponse(H,x)
plotResponse(H,x,xref)
plotResponse(H,x,coeff)
plotResponse( ___ ,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,x)` plots the range-Doppler response of the input signal, `x`, in decibels. This syntax is available when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `false`.

`plotResponse(H,x,xref)` plots the range-Doppler response after performing a dechirp operation on `x` using the reference signal, `xref`. This syntax is available when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `true`.

`plotResponse(H,x,coeff)` plots the range-Doppler response after performing a matched filter operation on `x` using the matched filter coefficients in `coeff`. This syntax is available when you set the `RangeMethod` property to `'Matched filter'`.

`plotResponse( ___ ,Name,Value)` plots the angle-Doppler response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns the handle of the image in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Range-Doppler response object.

x

Input data. Specific requirements depend on the syntax:

- In the syntax `plotResponse(H,x)`, each column of the matrix `x` represents a dechirped signal from one frequency sweep. The function assumes all sweeps in `x` are consecutive.
- In the syntax `plotResponse(H,x,xref)`, each column of the matrix `x` represents a signal from one frequency sweep. The function assumes all sweeps in `x` are consecutive and have not been dechirped yet.

- In the syntax `plotResponse(H,x,coeff)`, each column of the matrix `x` represents a signal from one pulse. The function assumes all pulses in `x` are consecutive.

In the case of an FMCW waveform with a triangle sweep, the sweeps alternate between positive and negative slopes. However, `phased.RangeDopplerResponse` is designed to process consecutive sweeps of the same slope. To apply `phased.RangeDopplerResponse` for a triangle-sweep system, use one of the following approaches:

- Specify a positive `SweepSlope` property value, with `x` corresponding to upsweeps only. In the plot, change the tick mark labels on the horizontal axis to reflect that the Doppler or speed values are half of what the plot shows by default.
- Specify a negative `SweepSlope` property value, with `x` corresponding to downsweeps only. In the plot, change the tick mark labels on the horizontal axis to reflect that the Doppler or speed values are half of what the plot shows by default.

You can specify this argument as single or double precision.

`xref`

Reference signal, specified as a column vector having the same number of rows as `x`. You can specify this argument as single or double precision.

`coeff`

Matched filter coefficients, specified as a column vector. You can specify this argument as single or double precision.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`NormalizeDoppler`

Set this value to `true` to normalize the Doppler frequency. Set this value to `false` to plot the range-Doppler response without normalizing the Doppler frequency. This parameter applies when you set the `DopplerOutput` property of `H` to `'Frequency'`.

Default: `false`

`Unit`

The unit of the plot. Valid values are `'db'`, `'mag'`, and `'pow'`.

Default: `'db'`

Examples

Range-Doppler Response of FMCW Signal

Compute the range-Doppler response of an FMCW signal using an FFT.

Load data for an FMCW signal that has not been dechirped. The signal contains the return from a target about 2200 m away. The signal has a normalized Doppler frequency of approximately -0.36 relative to the radar.

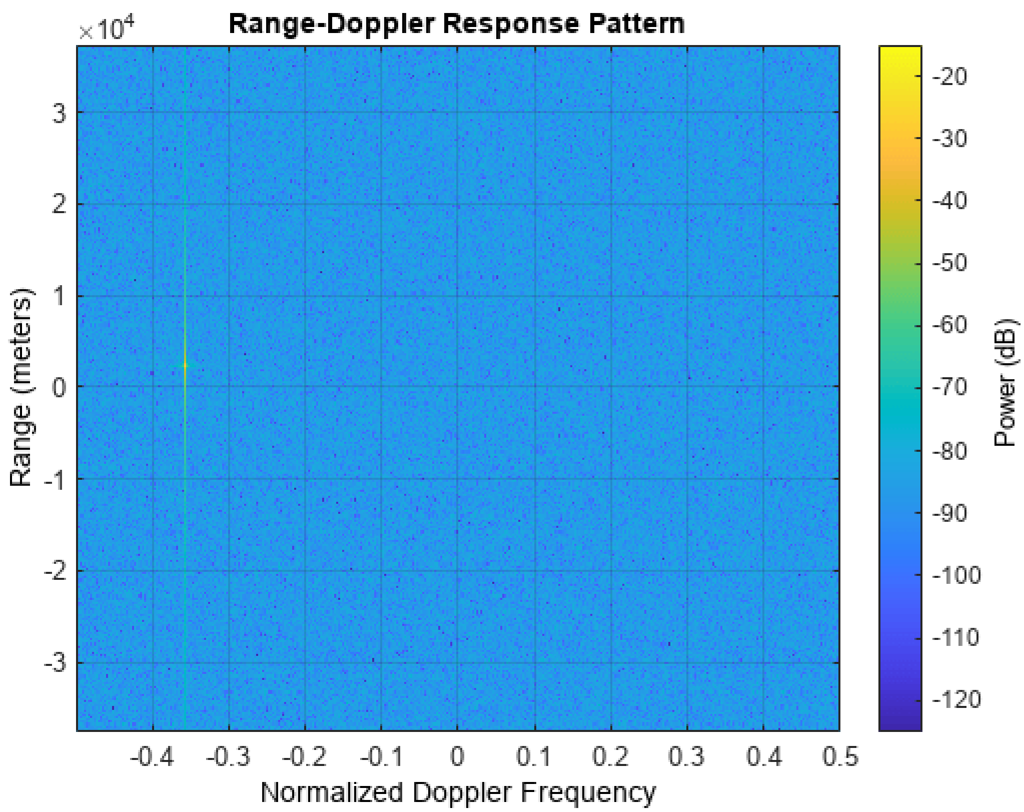
```
load RangeDopplerExampleData;
```

Create a range-Doppler response object.

```
hrdresp = phased.RangeDopplerResponse(...
   'RangeMethod','FFT',...
   'PropagationSpeed',RangeDopplerEx_Dechirp_PropSpeed,...
   'SampleRate',RangeDopplerEx_Dechirp_Fs,...
   'DechirpInput',true,...
   'SweepSlope',RangeDopplerEx_Dechirp_SweepSlope);
```

Plot the range-Doppler response.

```
plotResponse(hrdresp,...
   RangeDopplerEx_Dechirp_X,RangeDopplerEx_Dechirp_Xref,...
   'Unit','db','NormalizeDoppler',true)
```



See Also

Topics
"Automotive Adaptive Cruise Control Using FMCW Technology" (Radar Toolbox)

step

System object: `phased.RangeDopplerResponse`
Package: `phased`

Calculate range-Doppler response

Syntax

```
[resp,rnggrid,dopgrid] = step(H,x)
[resp,rnggrid,dopgrid] = step(H,x,xref)
[resp,rnggrid,dopgrid] = step(H,x,coeff)
[resp,rnggrid,dopgrid] = step(H, ___ ,prf)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[resp,rnggrid,dopgrid] = step(H,x)` calculates the range-Doppler response of the input signal, `x`. `resp` is the complex range-Doppler response. `rnggrid` and `dopgrid` provide the range samples and Doppler samples, respectively, at which the range-Doppler response is evaluated. This syntax is available when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `false`. This syntax is most commonly used with FMCW signals.

`[resp,rnggrid,dopgrid] = step(H,x,xref)` uses `xref` as the reference signal to dechirp `x`. This syntax is available when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `true`. This syntax is most commonly used with FMCW signals, where the reference signal is typically the transmitted signal.

`[resp,rnggrid,dopgrid] = step(H,x,coeff)` uses `coeff` as the matched filter coefficients. This syntax is available when you set the `RangeMethod` property to `'Matched filter'`. This syntax is most commonly used with pulsed signals, where the matched filter is the time reverse of the transmitted signal.

`[resp,rnggrid,dopgrid] = step(H, ___ ,prf)` uses `prf` as the pulse repetition frequency. These syntaxes are available when you set the `PRFSource` property to `'Input port'`. This syntax is most commonly used with pulsed signals, where the matched filter is the time reverse of the transmitted signal.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Range-Doppler response System object

x

Input data, specified as a complex-valued *K*-by-*L* matrix or *K*-by-*N*-by-*L* array where

- *K* denotes the number of fast-time samples.
- *N* denotes the number of channels such as beams or sensors. When *N* is one, only a single data channel is present.
- *L* denotes the number of pulses for matched-filter processing and the number of sweeps for FFT processing.

Specific requirements depend on the syntax:

- In the syntax `step(H,x)`, each column of `x` represents a dechirped signal from one frequency sweep. The function assumes all sweeps in `x` are consecutive.
- In the syntax `step(H,x,xref)`, each column of `x` represents a signal from one frequency sweep. The function assumes all sweeps in `x` are consecutive and are not dechirped.
- In the syntax `step(H,x,coeff)`, each column of the matrix `x` represents a signal from one pulse. The function assumes all pulses in `x` are consecutive.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

In the case of an FMCW waveform with a triangle sweep, the sweeps alternate between positive and negative slopes. However, `phased.RangeDopplerResponse` is designed to process consecutive sweeps of the same slope. To apply `phased.RangeDopplerResponse` for a triangle-sweep system, use one of the following approaches:

- Specify a positive `SweepSlope` property value, with `x` corresponding to upsweeps only. After obtaining the Doppler or speed values, divide them by 2.
- Specify a negative `SweepSlope` property value, with `x` corresponding to downsweeps only. After obtaining the Doppler or speed values, divide them by 2.

You can specify this argument as single or double precision.

xref

Reference signal, specified as a column vector having the same number of rows as `x`. You can specify this argument as single or double precision.

coeff

Matched filter coefficients, specified as a column vector. You can specify this argument as single or double precision.

prf

Pulse repetition frequency, specified as a positive scalar. `prf` must be less than or equal to the sample rate specified in the `SampleRate` property divided by the length of the first dimension of the input signal, `x`. You can specify this argument as single or double precision.

To enable this argument, set the `PRFSource` property to `'Input port'`.

Output Arguments

resp

Range-Doppler response of `x`, returned as a complex-valued *M*-by-*P* matrix or a *M*-by-*N*-by-*P* array. The values of *P* and *M* depend on the syntax. *N* has the same value as for the input argument, `x`.

Syntax	Values of *M* and *P*
`step(H,x)`	If you set the `RangeFFTLength` property to `'Auto'`, $M = K$, the length of the first dimension of `x`. Otherwise, *M* equals the value of the `RangeFFTLength` property. If you set the `DopplerFFTLength` property to `'Auto'`, $P = L$, the length of the last dimension of `x`. Otherwise, *P* equals the value of the `DopplerFFTLength` property.
`step(H,x,xref)`	*M* is the quotient of the length of the first dimension of `x` divided by the value of the `DecimationFactor` property. If you set the `DopplerFFTLength` property to `'Auto'`, $P = L$, the length of the last dimension of `x`. Otherwise, *P* equals the value of the `DopplerFFTLength` property.
`step(H,x,coeff)`	*M* is the number of rows of `x`. If you set the `DopplerFFTLength` property to `'Auto'`, $P = L$, the length of the last dimension of `x`. Otherwise, *P* equals the value of the `DopplerFFTLength` property.

rnggrid

Range samples at which the range-Doppler response is evaluated. `rnggrid` is a column vector of length *M*.

dopgrid

Doppler samples or speed samples at which the range-Doppler response is evaluated. `dopgrid` is a column vector of length *P*. Whether `dopgrid` contains Doppler or speed samples depends on the `DopplerOutput` property of `H`.

Examples

Range-Doppler Response Using Matched Filter

Compute the range-doppler response of a pulsed radar signal using a matched filter.

Load data for a pulsed radar signal. The signal includes three target returns. Two targets are approximately 2000 m away, while the third is approximately 3500 m away. In addition, two of the targets are stationary relative to the radar. The third is moving away from the radar at about 100 m/s.

```
load RangeDopplerExampleData;
```

Create a range-Doppler response object.

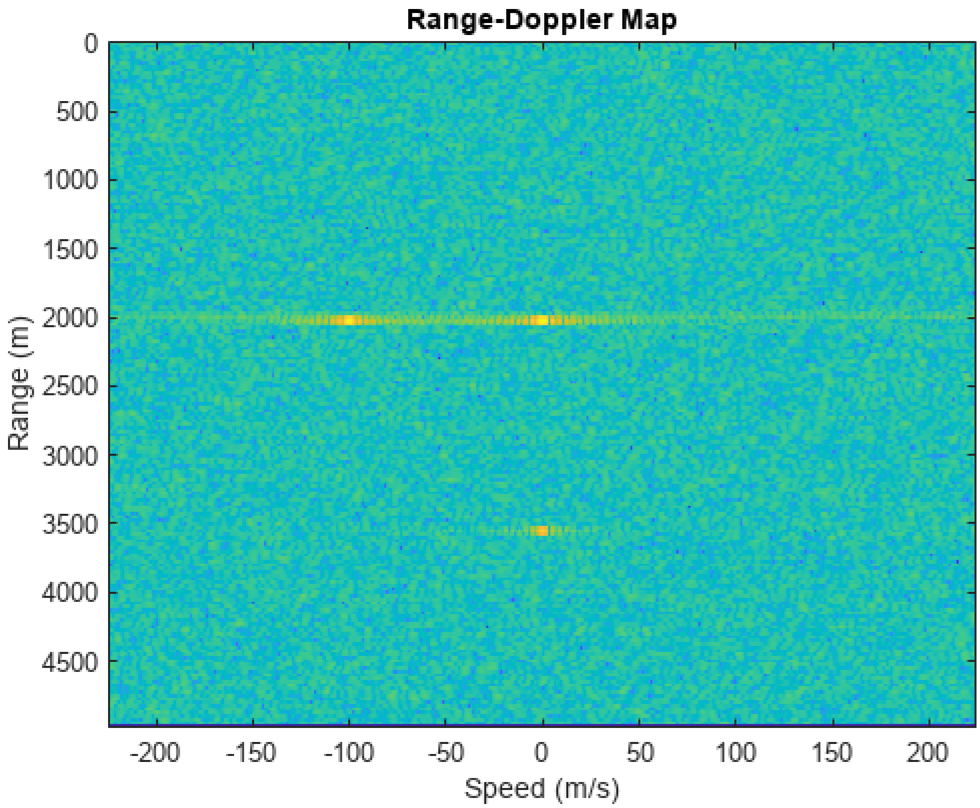
```
response = phased.RangeDopplerResponse('DopplerFFTLengthSource','Property', ...
   'DopplerFFTLength',RangeDopplerEx_MF_NFFTDOP, ...
   'SampleRate',RangeDopplerEx_MF_Fs,'DopplerOutput','Speed', ...
   'OperatingFrequency',RangeDopplerEx_MF_Fc);
```

Calculate the range-Doppler response.

```
[resp,rng_grid,dop_grid] = response(RangeDopplerEx_MF_X, ...
    RangeDopplerEx_MF_Coeff);
```

Plot the range-Doppler response.

```
imagesc(dop_grid,rng_grid,mag2db(abs(resp)));
xlabel('Speed (m/s)');
ylabel('Range (m)');
title('Range-Doppler Map');
```



Estimate Doppler and Range from Range-Doppler Response

Estimate the Doppler and range values of a single target from the range-Doppler response.

Load data for an FMCW signal that has not yet been dechirped. The signal contains the return from one target.

```
load RangeDopplerExampleData;
```

Create a range-Doppler response object.

```
hrdresp = phased.RangeDopplerResponse(...
   'RangeMethod','FFT',...
   'PropagationSpeed',RangeDopplerEx_Dechirp_PropSpeed,...
   'SampleRate',RangeDopplerEx_Dechirp_Fs,...
   'DechirpInput',true,...
   'SweepSlope',RangeDopplerEx_Dechirp_SweepSlope);
```

Obtain the range-Doppler response data.

```
[resp,rng_grid,dop_grid] = step(hrdresp,...
   RangeDopplerEx_Dechirp_X,RangeDopplerEx_Dechirp_Xref);
```

Estimate the range and Doppler by finding the location of the maximum response.

```
[x_temp,idx_temp] = max(abs(resp));
[~,dop_idx] = max(x_temp);
rng_idx = idx_temp(dop_idx);
dop_est = dop_grid(dop_idx)
```

```
dop_est = -712.8906
```

```
rng_est = rng_grid(rng_idx)
```

```
rng_est = 2250
```

The target is approximately 2250 meters away, and is moving fast enough to cause a Doppler shift of approximately -713 Hz.

phased.RangeEstimator

Package: phased

Range estimation

Description

The phased.RangeEstimator System object estimates the ranges of targets. Input to the estimator consists of a range-response or range-Doppler response data cube, and detection locations from a detector. When information about clusters of detections is available, the ranges are computed using cluster information. Clustering associates multiple detections into one extended detection.

To compute the detections for a range-response or range-Doppler cube:

1. Define and set up a range estimator using the “Construction” on page 1-1354 procedure that follows.
2. Call the step method to compute the range, using the properties you specify for the phased.RangeEstimator System object.

Note Instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

estimator = phased.RangeEstimator creates a range estimator System object, estimator.

estimator = phased.RangeEstimator(Name,Value) creates a System object, estimator, with each specified property Name set to the specified Value. You can specify additional name and value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

NumEstimatesSource — Source of number of range estimates to report
'Auto' (default) | 'Property'

Source of the number of range estimates to report, specified as 'Auto' or 'Property'.

If you set this property to 'Auto', the number of reported estimates is determined from the number of columns in the detidx input to the step method. If cluster IDs are provided, the number of estimates is determined from the number of unique cluster IDs in the clusterids input to the step method.

If you set this property to 'Property', the number of reported estimates is obtained from the value of the NumEstimates property.

Data Types: char

NumEstimates — Maximum number of estimates
1 (default) | positive integer

The maximum number of range estimates to report, specified as a positive integer. The number of requested estimates can be greater than the number of columns in the detidx argument or the number of unique IDs in the clusterids argument of the step method. In that case, the remainder is filled with NaN.

Dependencies

To enable this property, set the NumEstimatesSource property to 'Property'.

Data Types: single | double

ClusterInputPort — Accept cluster IDs as input
false (default) | true

Option to accept cluster IDs as an input argument to the step method, specified as false or true. Setting this property to true enables the clusterids input argument.

Data Types: logical

VarianceOutputPort — Output variance for range estimates
false (default) | true

Option to enable output of range estimate variances, specified as false or true. Range variances are returned by the rngvar output argument of the step method.

Data Types: logical

RMSResolution — Root-mean-square range resolution
1.0 (default) | positive scalar

Root-mean-square range resolution of the detection, specified as a positive scalar. The value of the RMSResolution must have the same units as the rangegrid input argument of the step method.

Dependencies

To enable this property, set the value of the VarianceOutputPort property to true.

Data Types: single | double

NoisePowerSource — Source of noise power values
'Property' (default) | 'Input port'

Source of noise power values, specified as 'Property' or 'Input port'. Noise power is used to compute range estimation variance and SNR. If you set this property to 'Property', the value of the NoisePower property represents the noise power at the detection locations. If you set this property to 'Input port', you can specify noise power using the noisepower input argument, of the step method.

Data Types: char

NoisePower — Noise power
1.0 (default) | positive scalar

Constant noise power value over the range-response or range-Doppler response data cube, specified as a positive real scalar. Noise power units are linear. The same noise power value is applied to all detections.

Dependencies

To enable this property, set the value of the `VarianceOutputPort` property to `true` and set `NoisePowerSource` to `'Property'`.

Data Types: `single` | `double`

Methods

step	Estimate target range

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Estimate Range and Speed of Three Targets

To estimate the range and speed of three targets, create a range-Doppler map using the `phased.RangeDopplerResponse` System object™. Then use the `phased.RangeEstimator` and `phased.DopplerEstimator` System objects to estimate range and speed. The transmitter and receiver are collocated isotropic antenna elements forming a monostatic radar system.

The transmitted signal is a linear FM waveform with a pulse repetition interval (PRI) of 7.0 μs and a duty cycle of 2%. The operating frequency is 77 GHz and the sample rate is 150 MHz.

```
fs = 150e6;
c = physconst('LightSpeed');
fc = 77.0e9;
pri = 7e-6;
prf = 1/pri;
```

Set up the scenario parameters. The transmitter and receiver are stationary and located at the origin. The targets are 500, 530, and 750 meters from the radar along the *x*-axis. The targets move along the *x*-axis at speeds of –60, 20, and 40 m/s. All three targets have a nonfluctuating radar cross-section (RCS) of 10 dB. Create the target and radar platforms.

```
Numtgts = 3;
tgtpos = zeros(Numtgts);
tgtpos(1,:) = [500 530 750];
tgtvel = zeros(3,Numtgts);
tgtvel(1,:) = [-60 20 40];
tgtrcs = db2pow(10)*[1 1 1];
tgtmotion = phased.Platform(tgtpos,tgtvel);
target = phased.RadarTarget('PropagationSpeed',c,'OperatingFrequency',fc, ...
    'MeanRCS',tgtrcs);
radarpos = [0;0;0];
radarvel = [0;0;0];
radarmotion = phased.Platform(radarpos,radarvel);
```

Create the transmitter and receiver antennas.

```
txantenna = phased.IsotropicAntennaElement;
rxantenna = clone(txantenna);
```

Set up the transmitter-end signal processing. Create an upsweep linear FM signal with a bandwidth of one half the sample rate. Find the length of the PRI in samples and then estimate the rms bandwidth and range resolution.

```
bw = fs/2;
waveform = phased.LinearFMWaveform('SampleRate',fs, ...
    'PRF',prf,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',fs/2, ...
    'DurationSpecification','Duty cycle','DutyCycle',0.02);
sig = waveform();
Nr = length(sig);
bwrms = bandwidth(waveform)/sqrt(12);
rngrms = c/bwrms;
```

Set up the transmitter and radiator System object properties. The peak output power is 10 W and the transmitter gain is 36 dB.

```
peakpower = 10;
txgain = 36.0;
transmitter = phased.Transmitter( ...
    'PeakPower',peakpower, ...
    'Gain',txgain, ...
    'InUseOutputPort',true);
radiator = phased.Radiator( ...
    'Sensor',txantenna,...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc);
```

Set up the free-space channel in two-way propagation mode.

```
channel = phased.FreeSpace( ...
    'SampleRate',fs, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc, ...
    'TwoWayPropagation',true);
```

Set up the receiver-end processing. Set the receiver gain and noise figure.

```
collector = phased.Collector( ...
    'Sensor',rxantenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc);
rxgain = 42.0;
noisefig = 1;
receiver = phased.ReceiverPreamp( ...
    'SampleRate',fs, ...
    'Gain',rxgain, ...
    'NoiseFigure',noisefig);
```

Loop over the pulses to create a data cube of 128 pulses. For each step of the loop, move the target and propagate the signal. Then put the received signal into the data cube. The data cube contains the received signal per pulse. Ordinarily, a data cube has three dimensions where the last dimension corresponds to antennas or beams. Because only one sensor is used, the cube has only two dimensions.

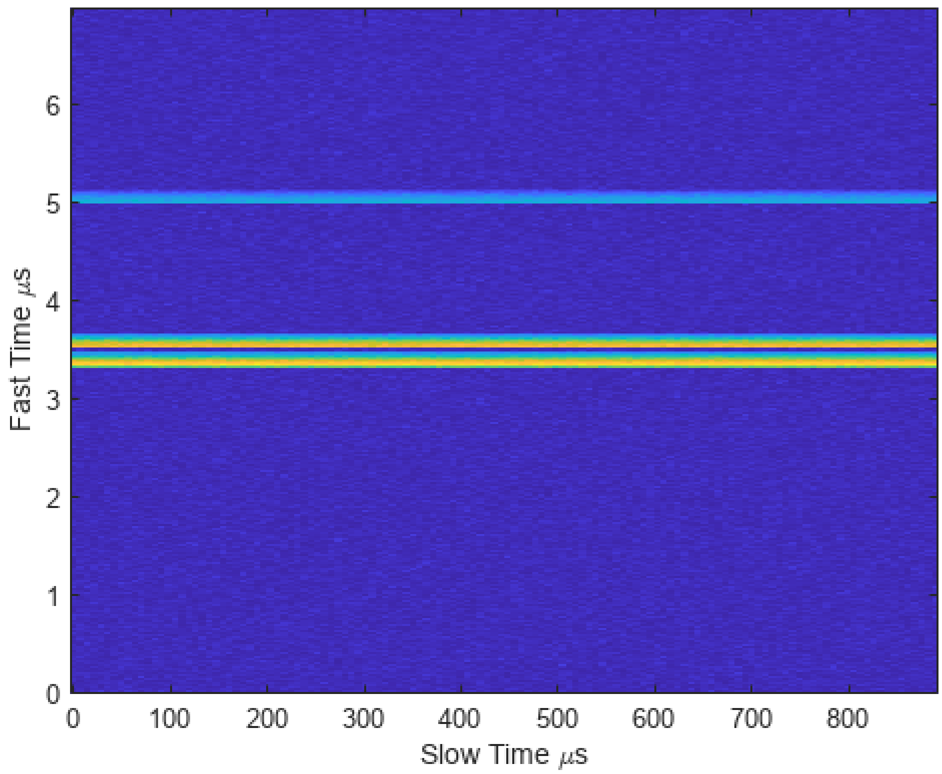
The processing steps are:

1. Move the radar and targets.
2. Transmit a waveform.
3. Propagate the waveform signal to the target.
4. Reflect the signal from the target.
5. Propagate the waveform back to the radar. Two-way propagation enables you to combine the return propagation with the outbound propagation.
6. Receive the signal at the radar.
7. Load the signal into the data cube.

```
Np = 128;
dt = pri;
cube = zeros(Nr,Np);
for n = 1:Np
    [sensorpos,sensorvel] = radarmotion(dt);
    [tgtpos,tgtvel] = tgtmotion(dt);
    [tgtrng,tgtang] = rangeangle(tgtpos,sensorpos);
    sig = waveform();
    [txsig,txstatus] = transmitter(sig);
    txsig = radiator(txsig,tgtang);
    txsig = channel(txsig,sensorpos,tgtpos,sensorvel,tgtvel);
    tgtsig = target(txsig);
    rxcol = collector(tgtsig,tgtang);
    rxsig = receiver(rxcol);
    cube(:,n) = rxsig;
end
```

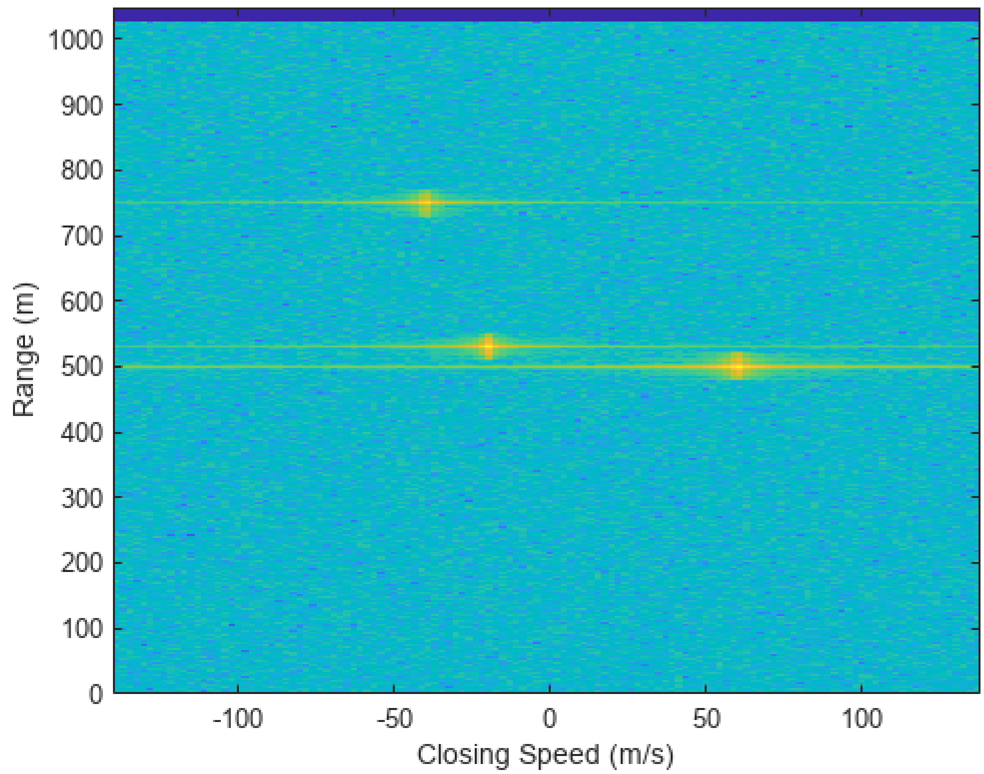
Display the data cube containing signals per pulse.

```
imagesc([0:(Np-1)]*pri*1e6,[0:(Nr-1)]/fs*1e6,abs(cube))
xlabel('Slow Time {\mu}s')
ylabel('Fast Time {\mu}s')
axis xy
```



Create and display the range-Doppler image for 128 Doppler bins. The image shows range vertically and speed horizontally. Use the linear FM waveform for match filtering. The image is here is the range-Doppler map.

```
ndop = 128;
rangedopresp = phased.RangeDopplerResponse('SampleRate',fs, ...
    'PropagationSpeed',c,'DopplerFFTLengthSource','Property', ...
    'DopplerFFTLength',ndop,'DopplerOutput','Speed', ...
    'OperatingFrequency',fc);
matchingcoeff = getMatchedFilter(waveform);
[rngdopresp,rnggrid,dopgrid] = rangedopresp(cube,matchingcoeff);
imagesc(dopgrid,rnggrid,10*log10(abs(rngdopresp)))
xlabel('Closing Speed (m/s)')
ylabel('Range (m)')
axis xy
```



Because the targets lie along the positive *x*-axis, positive velocity in the global coordinate system corresponds to negative closing speed. Negative velocity in the global coordinate system corresponds to positive closing speed.

Estimate the noise power after matched filtering. Create a constant noise background image for simulation purposes.

```
mfgain = matchingcoeff'*matchingcoeff;
dopgain = Np;
noisebw = fs;
noisepower = noisepow(noisebw,receiver.NoiseFigure,receiver.ReferenceTemperature);
noisepowerprc = mfgain*dopgain*noisepower;
noise = noisepowerprc*ones(size(rngdopresp));
```

Create the range and Doppler estimator objects.

```
rangeestimator = phased.RangeEstimator('NumEstimatesSource','Auto', ...
    'VarianceOutputPort',true,'NoisePowerSource','Input port', ...
    'RMSResolution',rngrms);
dopestimator = phased.DopplerEstimator('VarianceOutputPort',true, ...
    'NoisePowerSource','Input port','NumPulses',Np);
```

Locate the target indices in the range-Doppler image. Instead of using a CFAR detector, for simplicity, use the known locations and speeds of the targets to obtain the corresponding index in the range-Doppler image.

```
detidx = NaN(2,Numtgts);
tgtrng = rangeangle(tgtpos,radarpos);
```

```
tgtspd = radialspeed(tgtpos,tgtvel,radarpos,radarvel);
tgtdop = 2*speed2dop(tgtspd,c/fc);
for m = 1:numel(tgtrng)
    [~,iMin] = min(abs(rnggrid-tgtrng(m)));
    detidx(1,m) = iMin;
    [~,iMin] = min(abs(dopgrid-tgtspd(m)));
    detidx(2,m) = iMin;
end
```

Find the noise power at the detection locations.

```
ind = sub2ind(size(noise),detidx(1,:),detidx(2,:));
```

Estimate the range and range variance at the detection locations. The estimated ranges agree with the postulated ranges.

```
[rngest,rngvar] = rangeestimator(rngdopresp,rnggrid,detidx,noise(ind))
```

```
rngest = 3×1

  499.7911
  529.8380
  750.0983


rngvar = 3×1
10^-4 ×

    0.0273
    0.0276
    0.2094
```

Estimate the speed and speed variance at the detection locations. The estimated speeds agree with the predicted speeds.

```
[spdest,spdvar] = dopestimator(rngdopresp,dopgrid,detidx,noise(ind))
```

```
spdest = 3×1

   60.5241
  -19.6167
  -39.5838


spdvar = 3×1
10^-5 ×

    0.0806
    0.0816
    0.6188
```

More About

Date Cube

One input to the range estimator is a response data cube. To create a response data cube, use the `phased.RangeDopplerResponse` or `phased.RangeResponse` System objects. The first dimension of the cube represents the range. Only the first dimension is used to estimate range. All other dimensions are ignored. To interpret the detection location, you must pass in the `rnggrid` vector corresponding to the range values along this dimension. See “Radar Data Cube Concept”.

Algorithms

Range Estimation

The `phased.RangeEstimator` System object estimates the range of a detection by following these steps:

1. Input a range-processed response data cube obtained from either the `phased.RangeResponse` or `phased.RangeDopplerResponse` System object. The first dimension of the cube represents the fast-time or equivalent range of the returned signal samples. Only this dimension is used to estimate detection range. All others are ignored.
2. Input a matrix of detection indices that specify the location of detections in the data cube. Each column denotes a separate detection. The row entries designate indices into the data cube. You can obtain detection indices as an output of the `phased.CFARDetector` or `phased.CFARDetector2D` detectors. To return these indices, set the `OutputFormat` property of either CFAR detector to `'Detection index'`.
3. Optionally input a row vector of cluster IDs. This vector is equal in length to the number of detections. Each element of this vector assigns an ID to a corresponding detection. To form clusters of detections, the same ID can be assigned to more than one detection. To enable this option, set the `ClusterInputPort` property to `true`.
4. When `ClusterInputPort` is `false`, the object computes the range for each detection. The algorithm finds the response values at the detection location and at two adjacent indices in the cube along the range dimension. Then, the algorithm fits a quadratic curve to the magnitudes of the range response at these three locations and finds the location of the peak. When detections occur at the first or last sample in the range dimension, the range response is estimated from a two-point centroid. The estimation is at the location of the detection index and at the sample adjacent to the detection index.

   When `ClusterInputPort` is `true`, the object computes range for each cluster. The algorithm finds the indices of the largest response value in the cluster and fits a quadratic formula to that detection in the same way as for individual detections.
5. Convert the fractional index values of the fitted peak locations to range. To convert the indices, choose appropriate units for the `rnggrid` input argument of the `step` method. You can use values for `rnggrid` obtained from either the `phased.RangeResponse` or `phased.RangeDopplerResponse` System objects.

The object computes the estimated range variance using the Ziv-Zakai bound.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2017a

References

[1] Richards, M. *Fundamentals of Radar Signal Processing*. 2nd ed. McGraw-Hill Professional Engineering, 2014.

[2] Richards, M., J. Scheer, and W. Holm. *Principles of Modern Radar: Basic Principles*. SciTech Publishing, 2010.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

Functions
`rangeangle` | `bw2range`

Objects
`phased.DopplerEstimator` | `phased.RangeResponse` | `phased.RangeDopplerResponse` | `phased.CFARDetector` | `phased.CFARDetector2D`

Topics
"Radar Data Cube Concept"

step

System object: `phased.RangeEstimator`
Package: `phased`

Estimate target range

Syntax

```
rngest = step(estimator,resp,rnggrid,detidx)
[rngest,rngvar] = step(estimator,resp,rnggrid,detidx,noisepower)
[rngest,rngvar] = step(estimator,resp,rnggrid,detidx,clusterids)
[rngest,rngvar] = step(estimator,resp,rnggrid,detidx,noisepower,clusterids)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`rngest = step(estimator,resp,rnggrid,detidx)` estimates ranges from detections derived from the range response data cube, `resp`. Range estimates are computed for each detection position reported in `detidx`. The `rnggrid` argument sets the units for the range dimension of the response data cube.

`[rngest,rngvar] = step(estimator,resp,rnggrid,detidx,noisepower)` also specifies the noise power. This syntax applies when you set the `VarianceOutputPort` property to `true` and the `NoisePowerSource` property to `'Input port'`.

`[rngest,rngvar] = step(estimator,resp,rnggrid,detidx,clusterids)` also specifies the cluster IDs for the detections. This syntax applies when you set the `ClusterInputPort` to `true`.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example, `[rngest,rngvar] = step(estimator,resp,rnggrid,detidx,noisepower, clusterids)`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`estimator` — Range estimator
`phased.RangeEstimator` System object

Range estimator, specified as a `phased.RangeEstimator` System object.

Example: `phased.RangeEstimator`

`resp` — Range-processed response data cube
complex-valued M-by-1 column vector | complex-valued M-by-N matrix | complex-valued M-by-N-by-P matrix

Range-processed response data cube, specified as a complex-valued M-by-1 column vector, a complex-valued M-by-N matrix, or a complex-valued M-by-N-by-P array. M is the number of fast-time or range samples. N is the number of spatial elements, such as sensor elements or beams, P is the number of Doppler bins or pulses, depending on whether the data cube has been Doppler processed.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`

`rnggrid` — Range values along range dimension
real-valued M-by-1 column vector

Range values along range dimension of the `resp` argument, specified as a real-valued M-by-1 column vector. `rnggrid` defines the range values corresponding to the fast-time or range dimension. Range values must be monotonically increasing and equally spaced. Units are in meters.

Example: `[0.1,0.2,0.3]`

Data Types: `single` | `double`

`detidx` — Detection indices
real-valued N_d-by-Q matrix

Detection indices, specified as a real-valued N_d-by-Q matrix. Q is the number of detections and N_d is the number of dimensions of the response data cube, `resp`. Each column of `detidx` contains the N_d indices of the detection in the response data cube.

To generate detection indices, you can use the `phased.CFARDetector` object or `phased.CFARDetector2D` object.

Data Types: `single` | `double`

`noisepower` — Noise power at detection locations
positive scalar | real-valued 1-by-Q row vector of positive values

Noise power at detection locations, specified as a positive scalar or real-valued 1-by-Q row vector where Q is the number of detections specified in `detidx`.

Dependencies

To enable this input argument, set the value of the `NoisePowerSource` property to `'Input port'`.

Data Types: `single` | `double`

`clusterids` — Cluster IDs
real-valued 1-by-Q row vector of positive values

Cluster IDs, specified as a real-valued 1-by-*Q* row vector, where *Q* is the number of detections specified in `detidx`. Each element of `clusterids` corresponds to a column in `detidx`. Detections with the same cluster ID are in the same cluster.

Dependencies

To enable this input argument, set the value of the `ClusterInputPort` property to `true`.

Data Types: `single` | `double`

Output Arguments

`rngest` — Range estimates
real-valued *K*-by-1 column vector

Range estimates, returned as a real-valued *K*-by-1 column vector.

- When `ClusterInputPort` is `false`, Doppler estimates are computed for each detection location in the `detidx` argument. Then *K* equals the column dimension, *Q*, of `detidx`.
- When `ClusterInputPort` is `true`, Doppler estimates are computed for each cluster ID in the `clusterids` argument. Then *K* equals the number of unique cluster IDs, *Q*.

Data Types: `single` | `double`

`rngvar` — Range estimation variance
positive, real-valued *K*-by-1 column vector

Range estimation variance, returned as a positive, real-valued *K*-by-1 column vector, where *K* is the dimension of `rngest`. Each element of `rngvar` corresponds to an element of `rngest`. The estimator variance is computed using the Ziv-Zakai bound.

Data Types: `single` | `double`

Examples

Estimate Range and Speed of Three Targets

To estimate the range and speed of three targets, create a range-Doppler map using the `phased.RangeDopplerResponse` System object™. Then use the `phased.RangeEstimator` and `phased.DopplerEstimator` System objects to estimate range and speed. The transmitter and receiver are collocated isotropic antenna elements forming a monostatic radar system.

The transmitted signal is a linear FM waveform with a pulse repetition interval (PRI) of 7.0 μs and a duty cycle of 2%. The operating frequency is 77 GHz and the sample rate is 150 MHz.

```
fs = 150e6;
c = physconst('LightSpeed');
fc = 77.0e9;
pri = 7e-6;
prf = 1/pri;
```

Set up the scenario parameters. The transmitter and receiver are stationary and located at the origin. The targets are 500, 530, and 750 meters from the radar along the *x*-axis. The targets move along the *x*-axis at speeds of –60, 20, and 40 m/s. All three targets have a nonfluctuating radar cross-section (RCS) of 10 dB. Create the target and radar platforms.

```
Numtgts = 3;
tgtpos = zeros(Numtgts);
tgtpos(1,:) = [500 530 750];
tgtvel = zeros(3,Numtgts);
tgtvel(1,:) = [-60 20 40];
tgtrcs = db2pow(10)*[1 1 1];
tgtmotion = phased.Platform(tgtpos,tgtvel);
target = phased.RadarTarget('PropagationSpeed',c,'OperatingFrequency',fc, ...
    'MeanRCS',tgtrcs);
radarpos = [0;0;0];
radarvel = [0;0;0];
radarmotion = phased.Platform(radarpos,radarvel);
```

Create the transmitter and receiver antennas.

```
txantenna = phased.IsotropicAntennaElement;
rxantenna = clone(txantenna);
```

Set up the transmitter-end signal processing. Create an upsweep linear FM signal with a bandwidth of one half the sample rate. Find the length of the PRI in samples and then estimate the rms bandwidth and range resolution.

```
bw = fs/2;
waveform = phased.LinearFMWaveform('SampleRate',fs, ...
    'PRF',prf,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',fs/2, ...
    'DurationSpecification','Duty cycle','DutyCycle',0.02);
sig = waveform();
Nr = length(sig);
bwrms = bandwidth(waveform)/sqrt(12);
rngrms = c/bwrms;
```

Set up the transmitter and radiator System object properties. The peak output power is 10 W and the transmitter gain is 36 dB.

```
peakpower = 10;
txgain = 36.0;
transmitter = phased.Transmitter( ...
    'PeakPower',peakpower, ...
    'Gain',txgain, ...
    'InUseOutputPort',true);
radiator = phased.Radiator( ...
    'Sensor',txantenna,...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc);
```

Set up the free-space channel in two-way propagation mode.

```
channel = phased.FreeSpace( ...
    'SampleRate',fs, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc, ...
    'TwoWayPropagation',true);
```

Set up the receiver-end processing. Set the receiver gain and noise figure.

```
collector = phased.Collector( ...
    'Sensor',rxantenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc);
rxgain = 42.0;
noisefig = 1;
receiver = phased.ReceiverPreamp( ...
    'SampleRate',fs, ...
    'Gain',rxgain, ...
    'NoiseFigure',noisefig);
```

Loop over the pulses to create a data cube of 128 pulses. For each step of the loop, move the target and propagate the signal. Then put the received signal into the data cube. The data cube contains the received signal per pulse. Ordinarily, a data cube has three dimensions where the last dimension corresponds to antennas or beams. Because only one sensor is used, the cube has only two dimensions.

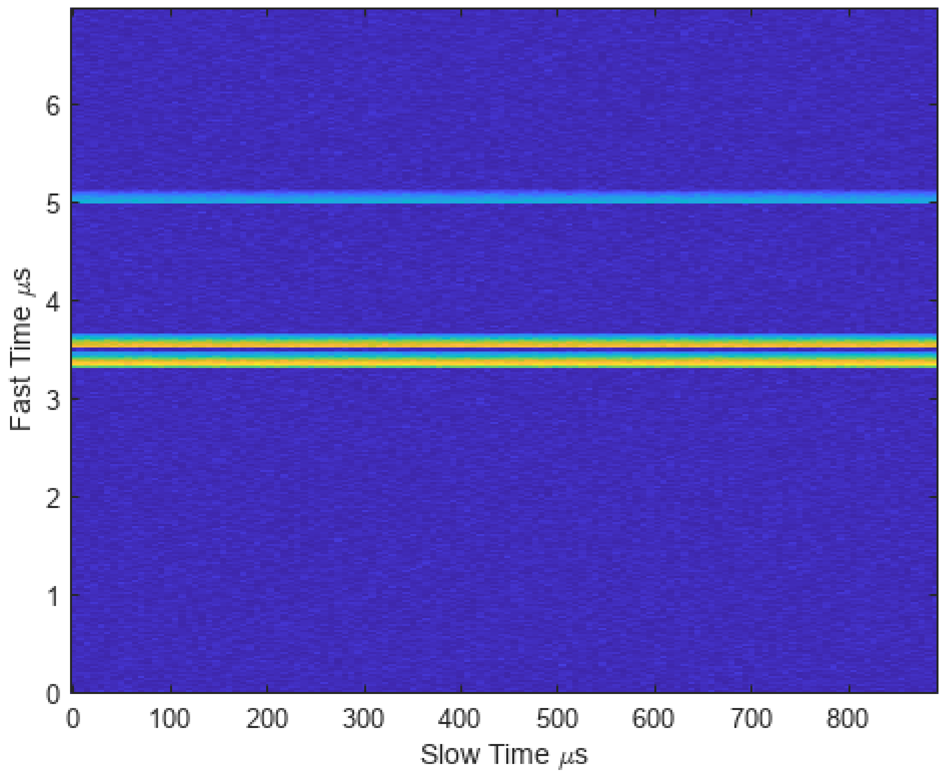
The processing steps are:

1. Move the radar and targets.
2. Transmit a waveform.
3. Propagate the waveform signal to the target.
4. Reflect the signal from the target.
5. Propagate the waveform back to the radar. Two-way propagation enables you to combine the return propagation with the outbound propagation.
6. Receive the signal at the radar.
7. Load the signal into the data cube.

```
Np = 128;
dt = pri;
cube = zeros(Nr,Np);
for n = 1:Np
    [sensorpos,sensorvel] = radarmotion(dt);
    [tgtpos,tgtvel] = tgtmotion(dt);
    [tgtrng,tgtang] = rangeangle(tgtpos,sensorpos);
    sig = waveform();
    [txsig,txstatus] = transmitter(sig);
    txsig = radiator(txsig,tgtang);
    txsig = channel(txsig,sensorpos,tgtpos,sensorvel,tgtvel);
    tgtsig = target(txsig);
    rxcol = collector(tgtsig,tgtang);
    rxsig = receiver(rxcol);
    cube(:,n) = rxsig;
end
```

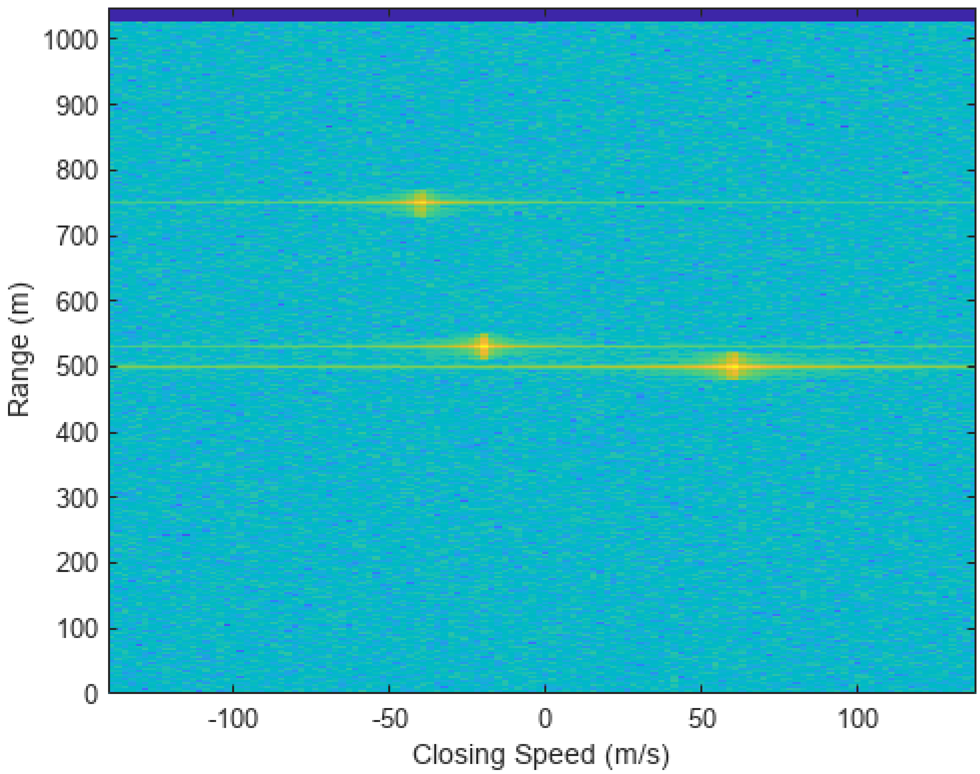
Display the data cube containing signals per pulse.

```
imagesc([0:(Np-1)]*pri*1e6,[0:(Nr-1)]/fs*1e6,abs(cube))
xlabel('Slow Time {\mu}s')
ylabel('Fast Time {\mu}s')
axis xy
```



Create and display the range-Doppler image for 128 Doppler bins. The image shows range vertically and speed horizontally. Use the linear FM waveform for match filtering. The image is here is the range-Doppler map.

```
ndop = 128;
rangedopresp = phased.RangeDopplerResponse('SampleRate',fs, ...
    'PropagationSpeed',c,'DopplerFFTLengthSource','Property', ...
    'DopplerFFTLength',ndop,'DopplerOutput','Speed', ...
    'OperatingFrequency',fc);
matchingcoeff = getMatchedFilter(waveform);
[rngdopresp,rnggrid,dopgrid] = rangedopresp(cube,matchingcoeff);
imagesc(dopgrid,rnggrid,10*log10(abs(rngdopresp)))
xlabel('Closing Speed (m/s)')
ylabel('Range (m)')
axis xy
```



Because the targets lie along the positive *x*-axis, positive velocity in the global coordinate system corresponds to negative closing speed. Negative velocity in the global coordinate system corresponds to positive closing speed.

Estimate the noise power after matched filtering. Create a constant noise background image for simulation purposes.

```
mfgain = matchingcoeff'*matchingcoeff;
dopgain = Np;
noisebw = fs;
noisepower = noisepow(noisebw,receiver.NoiseFigure,receiver.ReferenceTemperature);
noisepowerprc = mfgain*dopgain*noisepower;
noise = noisepowerprc*ones(size(rngdopresp));
```

Create the range and Doppler estimator objects.

```
rangeestimator = phased.RangeEstimator('NumEstimatesSource','Auto', ...
    'VarianceOutputPort',true,'NoisePowerSource','Input port', ...
    'RMSResolution',rngrms);
dopestimator = phased.DopplerEstimator('VarianceOutputPort',true, ...
    'NoisePowerSource','Input port','NumPulses',Np);
```

Locate the target indices in the range-Doppler image. Instead of using a CFAR detector, for simplicity, use the known locations and speeds of the targets to obtain the corresponding index in the range-Doppler image.

```
detidx = NaN(2,Numtgts);
tgtrng = rangeangle(tgtpos,radarpos);
```

```
tgtspd = radialspeed(tgtpos,tgtvel,radarpos,radarvel);
tgtdop = 2*speed2dop(tgtspd,c/fc);
for m = 1:numel(tgtrng)
    [~,iMin] = min(abs(rnggrid-tgtrng(m)));
    detidx(1,m) = iMin;
    [~,iMin] = min(abs(dopgrid-tgtspd(m)));
    detidx(2,m) = iMin;
end
```

Find the noise power at the detection locations.

```
ind = sub2ind(size(noise),detidx(1,:),detidx(2,:));
```

Estimate the range and range variance at the detection locations. The estimated ranges agree with the postulated ranges.

```
[rngest,rngvar] = rangeestimator(rngdopresp,rnggrid,detidx,noise(ind))
```

```
rngest = 3×1

  499.7911
  529.8380
  750.0983


rngvar = 3×1
10^-4 ×

    0.0273
    0.0276
    0.2094
```

Estimate the speed and speed variance at the detection locations. The estimated speeds agree with the predicted speeds.

```
[spdest,spdvar] = dopestimator(rngdopresp,dopgrid,detidx,noise(ind))
```

```
spdest = 3×1

   60.5241
  -19.6167
  -39.5838


spdvar = 3×1
10^-5 ×

    0.0806
    0.0816
    0.6188
```

phased.RangeResponse

Package: `phased`

Range response

Description

The `phased.RangeResponse` System object performs range filtering on fast-time (range) data, using either a matched filter or an FFT-based algorithm. The output is typically used as input to a detector. Matched filtering improves the SNR of pulsed waveforms. For continuous FM signals, FFT processing extracts the beat frequency of FMCW waveforms. Beat frequency is directly related to range.

The input to the range response object is a radar data cube. The organization of the data cube follows the Phased Array System Toolbox convention.

- The first dimension of the cube represents the fast-time samples or ranges of the received signals.
- The second dimension represents multiple spatial channels, such as different sensors or beams.
- The third dimension, slow-time, represent pulses.

Range filtering operates along the fast-time dimension of the cube. Processing along the other dimensions is not performed. If the data contains only one channel or pulse, the data cube can contain fewer than three dimensions. Because this object performs no Doppler processing, you can use the object to process noncoherent radar pulses.

The output of the range response object is also a data cube with the same number of dimensions as the input. Its first dimension contains range-processed data but its length can differ from the first dimension of the input data cube.

To compute the range response:

1. Define and set up your `phased.RangeResponse` System object. See "Construction" on page 1-1372.
2. Call the `step` method to compute the range response using the properties you specify for the `phased.RangeResponse` System object.

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`response = phased.RangeResponse` creates a range response System object, `response`.

`response = phased.RangeResponse(Name,Value)` creates a System object, `response`, with each specified property `Name` set to the specified `Value`. You can specify additional name and value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

RangeMethod — Range processing method
'Matched filter' (default) | 'FFT'

Range processing method, specified as 'Matched filter' or 'FFT'.

- 'Matched filter' — The object match-filters the incoming signal. This approach is commonly used for pulsed signals, where the matched filter is the time reverse of the transmitted signal.
- 'FFT' — The object applies an FFT to the input signal. This approach is commonly used for chirped signals such as FMCW and linear FM pulsed signals.

Example: 'Matched filter'

Data Types: char

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed').

Example: 3e8

Data Types: single | double

SampleRate — Signal sample rate
1e6 (default) | positive real-valued scalar

Signal sample rate, specified as a positive real-valued scalar. Units are in hertz.

Example: 1e6

Data Types: single | double

SweepSlope — Linear FM sweep slope
1.0e9 (default) | scalar

Linear FM sweep slope, specified as a scalar. The fast-time dimension of the signal input argument to step must correspond to sweeps having this slope.

Example: 1.5e9

Dependencies

To enable this property, set the RangeMethod property to 'FFT'.

Data Types: single | double

DechirpInput — Enable dechirping of input signals
false (default) | true

Option to enable dechirping of input signals, specified as false or true. Set this property to false to indicate that the input signal is already dechirped and no dechirp operation is necessary. Set this property to true when the input signal requires dechirping.

Dependencies

To enable this property, set the RangeMethod property to 'FFT'.

Data Types: `logical`

`DecimationFactor` — Decimation factor for dechirped signals
`1` (default) | positive integer

Decimation factor for dechirped signals, specified as a positive integer. The decimation algorithm uses a 30th-order FIR filter generated by `fir1(30,1/D)`, where `D` is the decimation factor. The default value of `1` implies no decimation.

When processing FMCW signals, decimating the dechirped signal is useful for reducing the load on A/D converters.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `true`.

Data Types: `single` | `double`

`RangeFFTLengthSource` — Source of FFT length for range processing of dechirped signals
`'Auto` (default) | `'Property'`

Source of the FFT length used for the range processing of dechirped signals, specified as `'Auto'` or `'Property'`.

- `'Auto'` — The FFT length equals the length of the fast-time dimension of the input data cube.
- `'Property'` — Specify the FFT length by using the `RangeFFTLength` property.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'`.

Data Types: `char`

`RangeFFTLength` — FFT length used for range processing
`1024` (default) | positive integer

FFT length used for range processing, specified as a positive integer.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeFFTLengthSource` property to `'Property'`

Data Types: `single` | `double`

`RangeWindow` — FFT weighting window for range processing
`'None'` (default) | `'Hamming'` | `'Chebyshev'` | `'Hann'` | `'Kaiser'` | `'Taylor'` | `'Custom'`

FFT weighting window for range processing, specified as `'None'`, `'Hamming'`, `'Chebyshev'`, `'Hann'`, `'Kaiser'`, `'Taylor'`, or `'Custom'`.

If you set this property to `'Taylor'`, the generated Taylor window has four nearly constant sidelobes next to the mainlobe.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'`.

Data Types: `char`

`RangeSidelobeAttenuation` — Sidelobe attenuation for range processing
`30` (default) | positive scalar

Sidelobe attenuation for range processing, specified as a positive scalar. Attenuation applies to Kaiser, Chebyshev, or Taylor windows. Units are in dB.

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeWindow` property to `'Kaiser'`, `'Chebyshev'`, or `'Taylor'`.

Data Types: `single` | `double`

`CustomRangeWindow` — Custom window for range processing
`@hamming` (default) | function handle | cell array

Custom window for range processing, specified as a function handle or a cell array containing a function handle as its first entry. If you do not specify a window length, the object computes the window length and passes that into the function. If you specify a cell array, the remaining cells of the array can contain arguments to the function. If you use only the function handle without passing in arguments, all arguments take their default values.

If you write your own window function, the first argument must be the length of the window.

Note Instead of using a cell array, you can pass in all arguments by constructing a handle to an anonymous function. For example, you can set the value of `CustomRangeWindow` to `@(n)taylorwin(n,nbar,sll)`, where you have previously set the values of `nbar` and `sll`.

Example: `{@taylor,5,-35}`

Dependencies

To enable this property, set the `RangeMethod` property to `'FFT'` and the `RangeWindow` property to `'Custom'`.

Data Types: `function_handle` | `cell`

`ReferenceRangeCentered` — Set reference range at center of range grid
`true` (default) | `false`

Set reference range at center of range grid, specified as `true` or `false`. Setting this property to `true` enables you to set the reference range at the center of the range grid. Setting this property to `false` sets the reference range to the beginning of the range grid.

Dependencies

To enable this property, set the `RangeMethod` to `'FFT'`.

Data Types: `logical`

`ReferenceRange` — Reference range of range grid
`0.0` (default) | nonnegative scalar

Reference range of the range grid, specified as a nonnegative scalar.

- If you set the `RangeMethod` property to `'Matched filter'`, the reference range is set to the start of the range grid.
- If you set the `RangeMethod` property to `'FFT'`, the reference range is determined by the `ReferenceRangeCentered` property.
  - When you set the `ReferenceRangeCentered` property to `true`, the reference range is set to the center of the range grid.
  - When you set the `ReferenceRangeCentered` property to `false`, the reference range is set to the start of the range grid.

  Units are in meters.

This property is tunable.

Example: `1000.0`

Data Types: `single` | `double`

`MaximumNumInputSamplesSource` — Source of maximum number of samples
`'Auto'` (default) | `'Property'`

The source of the maximum number of samples the input signal, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the object automatically allocates enough memory to buffer the first input signal. When you set this property to `'Property'`, you specify the maximum number of samples in the input signal using the `MaximumNumInputSamples` property. Any input signal longer than that value is truncated.

To use this object with variable-size input signals in a MATLAB Function Block in Simulink, set the `MaximumNumInputSamplesSource` property to `'Property'` and set a value for the `MaximumNumInputSamples` property.

Example: `'Property'`

`MaximumNumInputSamples` — Maximum number of input signal samples
`100` (default) | positive integer

Maximum number of samples in the input signal, specified as a positive integer. Any input signal longer than this value is truncated. The input signal is the first argument to the `step` method. The number of samples is the number of rows in the input.

Example: `2048`

Dependencies

To enable this property, set the `RangeMethod` property to `'Matched filter'` and set the `MaximumNumInputSamplesSource` property to `'Property'`.

Data Types: `single` | `double`

Methods

plotResponse	Plot range response
step	Range response

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Range Response of Three Targets

Compute the radar range response of three targets by using the `phased.RangeResponse` System object™. The transmitter and receiver are collocated isotropic antenna elements forming a monostatic radar system. The transmitted signal is a linear FM waveform with a pulse repetition interval of 7.0 μs and a duty cycle of 2%. The operating frequency is 77 GHz and the sample rate is 150 MHz.

```
fs = 150e6;
c = physconst('LightSpeed');
fc = 77e9;
pri = 7e-6;
prf = 1/pri;
```

Set up the scenario parameters. The radar transmitter and receiver are stationary and located at the origin. The targets are 500, 530, and 750 meters from the radars on the *x*-axis. The targets move along the *x*-axis at speeds of −60, 20, and 40 m/s. All three targets have a nonfluctuating radar cross-section (RCS) of 10 dB.

Create the target and radar platforms.

```
Numtgts = 3;
tgtpos = zeros(Numtgts);
tgtpos(1,:) = [500 530 750];
tgtvel = zeros(3,Numtgts);
tgtvel(1,:) = [-60 20 40];
tgtrcs = db2pow(10)*[1 1 1];
tgtmotion = phased.Platform(tgtpos,tgtvel);
target = phased.RadarTarget('PropagationSpeed',c,'OperatingFrequency',fc, ...
    'MeanRCS',tgtrcs);
radarpos = [0;0;0];
radarvel = [0;0;0];
radarmotion = phased.Platform(radarpos,radarvel);
```

Create the transmitter and receiver antennas.

```
txantenna = phased.IsotropicAntennaElement;
rxantenna = clone(txantenna);
```

Set up the transmitter-end signal processing. Create an upsweep linear FM signal with a bandwidth of half the sample rate. Find the length of the pri in samples and then estimate the rms bandwidth and range resolution.

```
bw = fs/2;
waveform = phased.LinearFMWaveform('SampleRate',fs, ...
    'PRF',prf,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',fs/2, ...
    'DurationSpecification','Duty cycle','DutyCycle',0.02);
sig = waveform();
Nr = length(sig);
```

```
bwrms = bandwidth(waveform)/sqrt(12);
rngrms = c/bwrms;
```

Set up the transmitter and radiator System object properties. The peak output power is 10 W and the transmitter gain is 36 dB.

```
peakpower = 10;
txgain = 36.0;
transmitter = phased.Transmitter( ...
    'PeakPower',peakpower, ...
    'Gain',txgain, ...
    'InUseOutputPort',true);
radiator = phased.Radiator( ...
    'Sensor',txantenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc);
```

Create a free-space propagation channel in two-way propagation mode.

```
channel = phased.FreeSpace( ...
    'SampleRate',fs, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc, ...
    'TwoWayPropagation',true);
```

Set up the receiver-end processing. The receiver gain is 42 dB and noise figure is 10.

```
collector = phased.Collector( ...
    'Sensor',rxantenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc);
rxgain = 42.0;
noisefig = 10;
receiver = phased.ReceiverPreamp( ...
    'SampleRate',fs, ...
    'Gain',rxgain, ...
    'NoiseFigure',noisefig);
```

Loop over 128 pulses to build a data cube. For each step of the loop, move the target and propagate the signal. Then put the received signal into the data cube. The data cube contains the received signal per pulse. Ordinarily, a data cube has three dimensions, where last dimension corresponds to antennas or beams. Because only one sensor is used in this example, the cube has only two dimensions.

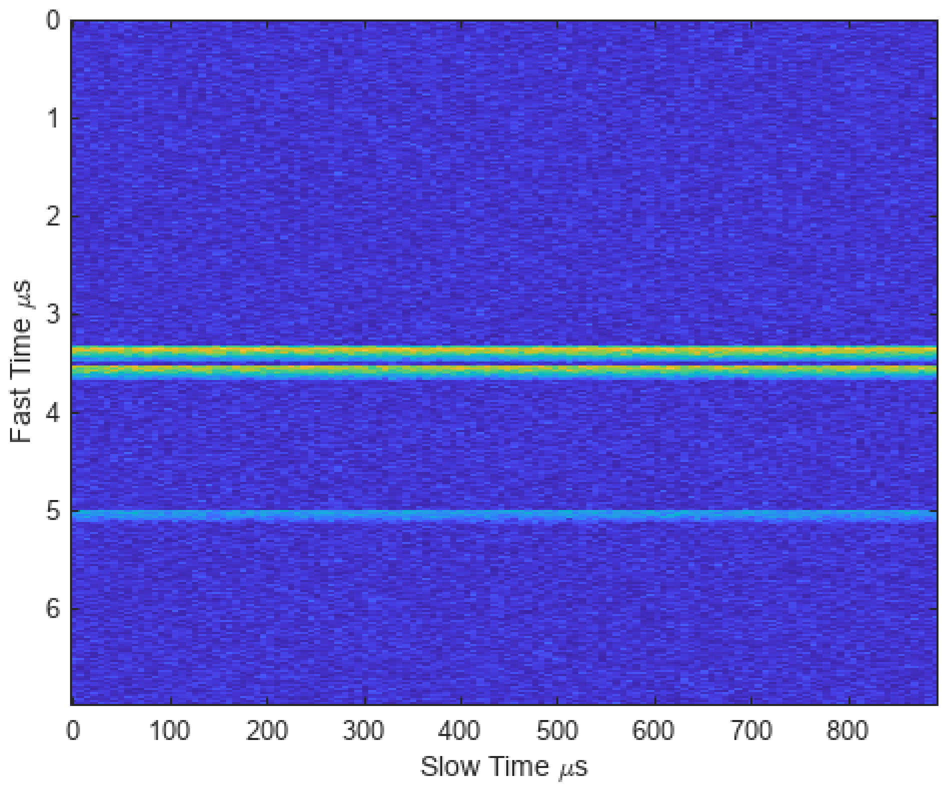
The processing steps are:

1. Move the radar and targets.
2. Transmit a waveform.
3. Propagate the waveform signal to the target.
4. Reflect the signal from the target.
5. Propagate the waveform back to the radar. Two-way propagation mode enables you to combine the returned propagation with the outbound propagation.
6. Receive the signal at the radar.
7. Load the signal into the data cube.

```
Np = 128;
cube = zeros(Nr,Np);
for n = 1:Np
    [sensorpos,sensorvel] = radarmotion(pri);
    [tgtpos,tgtvel] = tgtmotion(pri);
    [tgtrng,tgtang] = rangeangle(tgtpos,sensorpos);
    sig = waveform();
    [txsig,txstatus] = transmitter(sig);
    txsig = radiator(txsig,tgtang);
    txsig = channel(txsig,sensorpos,tgtpos,sensorvel,tgtvel);
    tgtsig = target(txsig);
    rxcol = collector(tgtsig,tgtang);
    rxsig = receiver(rxcol);
    cube(:,n) = rxsig;
end
```

Display the image of the data cube containing signals per pulse.

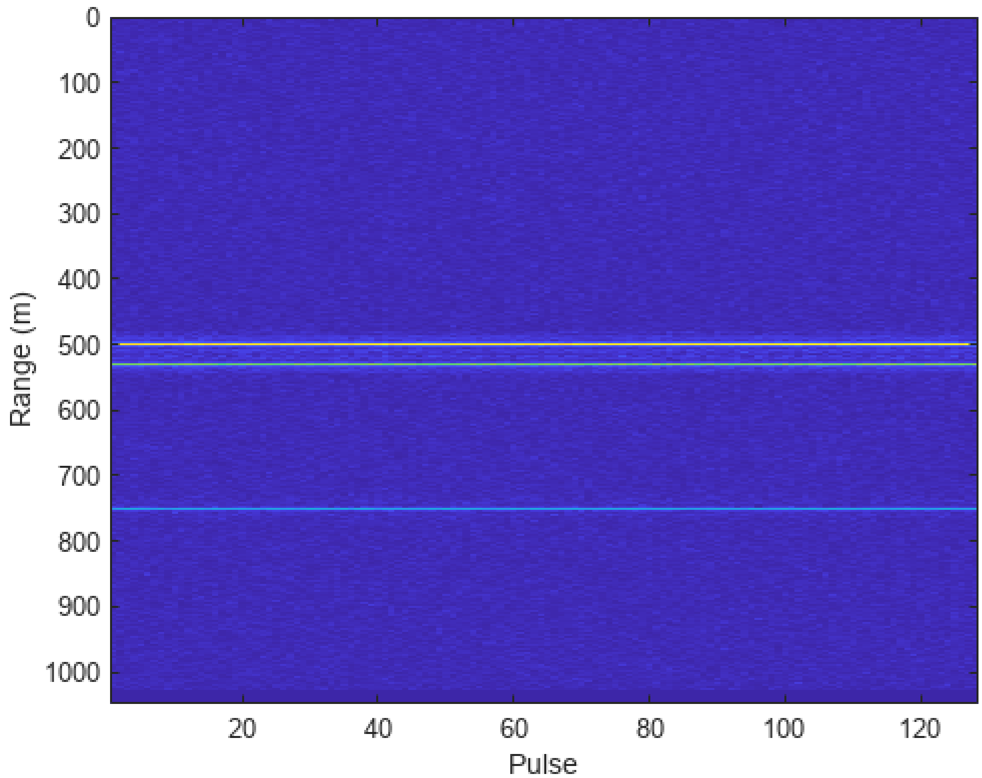
```
imagesc([0:(Np-1)]*pri*1e6,[0:(Nr-1)]/fs*1e6,abs(cube))
xlabel('Slow Time {\mu}s')
ylabel('Fast Time {\mu}s')
```



Create a `phased.RangeResponse` System object in matched filter mode. Then, display the range response image for the 128 pulses. The image shows range vertically and pulse number horizontally.

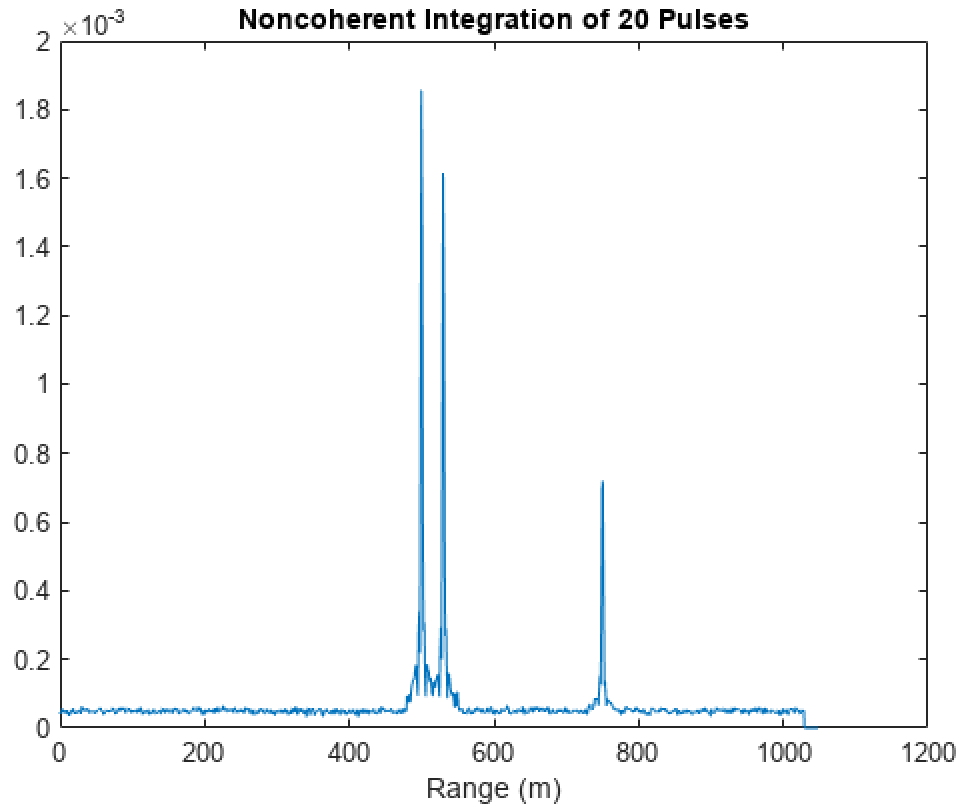
```
matchingcoeff = getMatchedFilter(waveform);
ndop = 128;
rangeresp = phased.RangeResponse('SampleRate',fs,'PropagationSpeed',c);
```

```
[resp,rnggrid] = rangeresp(cube,matchingcoeff);
imagesc([1:Np],rnggrid,abs(resp))
xlabel('Pulse')
ylabel('Range (m)')
```



Integrate 20 pulses noncoherently.

```
intpulse = pulsint(resp(:,1:20),'noncoherent');
plot(rnggrid,abs(intpulse))
xlabel('Range (m)')
title('Noncoherent Integration of 20 Pulses')
```



Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2017a

References

[1] Richards, M. *Fundamentals of Radar Signal Processing, 2nd ed*. McGraw-Hill Professional Engineering, 2014.

[2] Richards, M., J. Scheer, and W. Holm, *Principles of Modern Radar: Basic Principles*. SciTech Publishing, 2010.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `CustomRangeWindow` property is not supported.
- The `plotResponse` method is not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

Functions
`bw2range` | `fir1` | `chebwin` | `dechirp` | `hann` | `hamming` | `kaiser` | `taylorwin` | `rangeangle`

Objects
`phased.RangeAngleResponse` | `phased.RangeDopplerResponse` | `phased.AngleDopplerResponse` | `phased.MatchedFilter` | `phased.DopplerEstimator` | `phased.RangeEstimator` | `phased.CFARDetector` | `phased.CFARDetector2D`

Topics
“Radar Data Cube”

plotResponse

System object: `phased.RangeResponse`
Package: `phased`

Plot range response

Syntax

```
plotResponse(response,x)
plotResponse(response,x,xref)
plotResponse(response,x,coeff)
plotResponse(response, ___ ,Name,Value)
```

Description

`plotResponse(response,x)` plots the range response of a dechirped input signal, `x`. This syntax applies when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `false`.

`plotResponse(response,x,xref)` plots the range response `x`, after performing a dechirp operation using the reference signal, `xref`. This syntax applies when you set the `RangeMethod` property to `'FFT'` and the `DechirpInput` property to `true`.

`plotResponse(response,x,coeff)` plots the range response of `x` after match filtering using the match filter coefficients, `coeff`. This syntax applies when you set the `RangeMethod` property to `'Matched filter'`.

`plotResponse(response, ___ ,Name,Value)` plots the range response with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`response` — Range response
`phased.RangeResponse` System object

Range response, specified as a `phased.RangeResponse` System object.

Example: `phased.RangeResponse`

`x` — Input radar data cube
complex-valued *K*-element column vector | complex-valued *K*-by-*L* matrix | complex-valued *K*-by-*N*-by-*L* array

Input radar data cube, specified as a complex-valued *K*-by-1 column vector, a *K*-by-*L* matrix, or *K*-by-*N*-by-*L* array.

- *K* is the number of fast-time or range samples.
- *N* is the number of independent spatial channels such as sensors or directions.
- *L* is the slow-time dimension that corresponds to the number of pulses or sweeps in the input signal.

See “Radar Data Cube”.

Each *K*-element fast-time dimension is processed independently.

For FMCW waveforms with a triangle sweep, the sweeps alternate between positive and negative slopes. However, `phased.RangeResponse` is designed to process consecutive sweeps of the same slope. To apply `phased.RangeResponse` for a triangle-sweep system, use one of the following approaches:

- Specify a positive `SweepSlope` property value, with `x` corresponding to upsweeps only. After obtaining the Doppler or speed values, divide them by 2.
- Specify a negative `SweepSlope` property value, with `x` corresponding to downsweeps only. After obtaining the Doppler or speed values, divide them by 2.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`

`xref` — Reference signal used for dechirping
complex-valued *K*-by-1 column vector

Reference signal used for dechirping, specified as a complex-valued *K*-by-1 column vector. The value of *K* must equal the length of the first dimension of `x`.

Dependencies

To enable this input argument, set the value of `RangeMethod` to `'FFT'` and `DechirpInput` to `true`.

Data Types: `single` | `double`

`coeff` — Matched filter coefficients
complex-valued *P*-by-1 column vector

Matched filter coefficients, specified as a complex-valued *P*-by-1 column vector. *P* must be less than or equal to *K*, the length of the fast-time dimension.

Dependencies

To enable this input argument, set `RangeMethod` property to `'Matched filter'`.

Data Types: `single` | `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Unit` — Vertical axes units
`'db'` (default) | `'mag'` | `'pow'`

Units for vertical axis of plot, specified as `'db'`, `'mag'`, or `'pow'`.

Example: `'pow'`

Data Types: `char`

Examples

Plot Range Response of Three Targets

Plot the radar range response of three targets using the `plotResponse` method of the `phased.RangeResponse` System object™. The transmitter and receiver are collocated isotropic antenna elements forming a monostatic radar system. The transmitted signal is a linear FM waveform with a pulse repetition interval of 7.0 μs and a duty cycle of 2%. The operating frequency is 77 GHz and the sample rate is 150 MHz.

```
fs = 150e6;fs = 150e6;
c = physconst('LightSpeed');
fc = 77e9;
pri = 7e-6;
prf = 1/pri;
```

Set up the scenario parameters. The radar transmitter and receiver are stationary and located at the origin. The targets are 500, 530, and 750 meters from the radars on the *x*-axis. The targets move along the *x*-axis at speeds of −60, 20, and 40 m/s. All three targets have a nonfluctuating RCS of 10 dB.

Create the target and radar platforms.

```
Numtgts = 3;
tgtpos = zeros(Numtgts);
tgtpos(1,:) = [500 530 750];
tgtvel = zeros(3,Numtgts);
tgtvel(1,:) = [-60 20 40];
tgtrcs = db2pow(10)*[1 1 1];
tgtmotion = phased.Platform(tgtpos,tgtvel);
target = phased.RadarTarget('PropagationSpeed',c,'OperatingFrequency',fc, ...
    'MeanRCS',tgtrcs);
radarpos = [0;0;0];
radarvel = [0;0;0];
radarmotion = phased.Platform(radarpos,radarvel);
```

Create the transmitter and receiver antennas.

```
txantenna = phased.IsotropicAntennaElement;
rxantenna = clone(txantenna);
```

Set up the transmitter-end signal processing. Construct an upsweep linear FM signal with a bandwidth of half the sample rate. Find the rms bandwidth and rms range resolution.

```
bw = fs/2;
waveform = phased.LinearFMWaveform('SampleRate',fs,...
    'PRF',prf,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',fs/2,...
    'DurationSpecification','Duty cycle','DutyCycle',.02);
sig = waveform();
Nr = length(sig);
bwrms = bandwidth(waveform)/sqrt(12);
rngrms = c/bwrms;
```

Set up the transmitter and radiator System object properties. The peak output power is 10 W and the transmitter gain is 36 dB.

```
peakpower = 10;
txgain = 36.0;
transmitter = phased.Transmitter(...
    'PeakPower',peakpower,...
    'Gain',txgain,...
    'InUseOutputPort',true);
radiator = phased.Radiator(...
    'Sensor',txantenna,...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc);
```

Create a free-space propagation channel in two-way propagation mode.

```
channel = phased.FreeSpace(...
    'SampleRate',fs,...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc,...
    'TwoWayPropagation',true);
```

Set up the receiver-end processing. The receiver gain is 42 dB and noise figure is 10.

```
collector = phased.Collector(...
    'Sensor',rxantenna,...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc);
rxgain = 42.0;
noisefig = 10;
receiver = phased.ReceiverPreamp(...
    'SampleRate',fs,...
    'Gain',rxgain,...
    'NoiseFigure',noisefig);
```

Loop over 128 pulses to build a data cube. For each step of the loop, move the target and propagate the signal. Then put the received signal into the data cube. The data cube contains the received signal per pulse. Ordinarily, a data cube has three dimensions. The last dimension corresponds to antennas or beams. Because only one sensor is used in this example, the cube has only two dimensions.

The processing steps are:

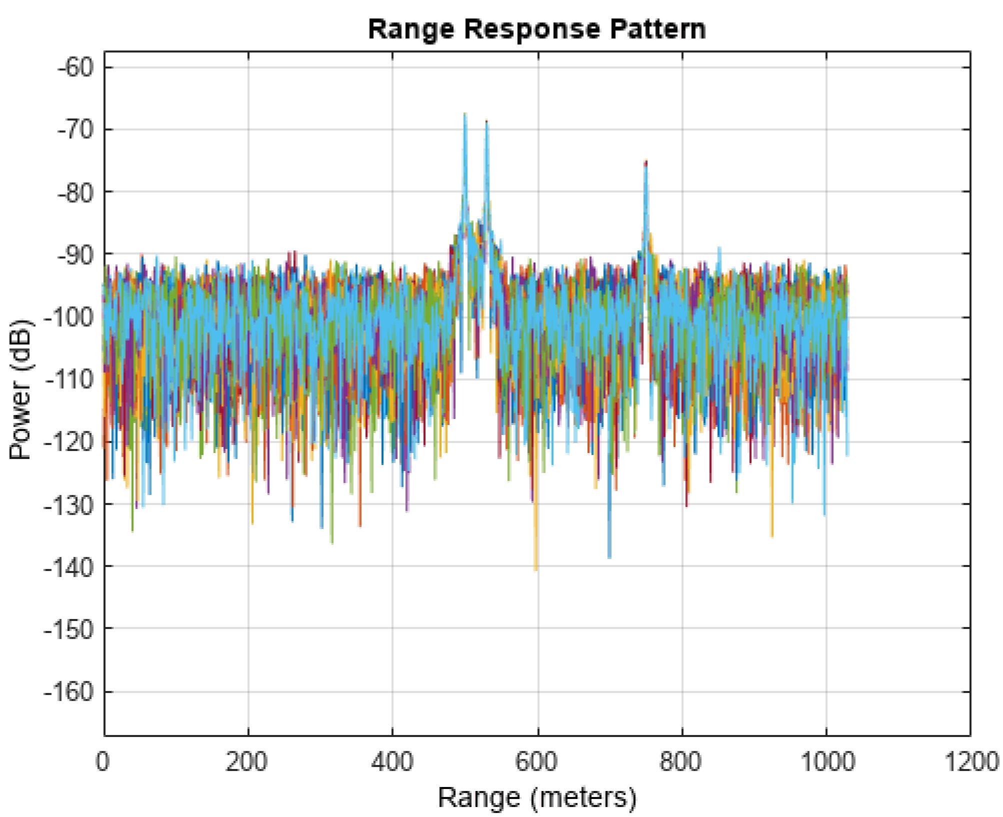
1. Move the radar and targets.
2. Transmit a waveform.
3. Propagate the waveform signal to the target.
4. Reflect the signal from the target.
5. Propagate the waveform back to the radar. Two-way propagation mode allows the return propagation to be combined with the outbound propagation.
6. Receive the signal at the radar.
7. Load the signal into the data cube.

```
Np = 128;
cube = zeros(Nr,Np);
for n = 1:Np
```

```
    [sensorpos,sensorvel] = radarmotion(pri);
    [tgtpos,tgtvel] = tgtmotion(pri);
    [tgtrng,tgtang] = rangeangle(tgtpos,sensorpos);
    sig = waveform();
    [txsig,txstatus] = transmitter(sig);
    txsig = radiator(txsig,tgtang);
    txsig = channel(txsig,sensorpos,tgtpos,sensorvel,tgtvel);
    tgtsig = target(txsig);
    rxcol = collector(tgtsig,tgtang);
    rxsig = receiver(rxcol);
    cube(:,n) = rxsig;
end
```

Create a `phased.RangeResponse` System object in matched filter mode. Then, call the `plotResponse` method to show the first 20 pulses.

```
matchcoeff = getMatchedFilter(waveform);
rangeresp = phased.RangeResponse('SampleRate',fs,'PropagationSpeed',c);
plotResponse(rangeresp,cube(:,1:20),matchcoeff);
```



Version History

Introduced in R2017a

step

System object: `phased.RangeResponse`
Package: `phased`

Range response

Syntax

```
[resp,rnggrid] = step(response,x)
[resp,rnggrid] = step(response,x,xref)
[resp,rnggrid] = step(response,x,coeff)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`[resp,rnggrid] = step(response,x)` computes the range response, `resp`, for the input signal, `x`, and the range values, rnggrid, corresponding to the response. This syntax applies when you set `RangeMethod` to `'FFT'` and `DechirpInput` to `false`. This syntax assumes that the input signal has already been dechirped. This syntax is most commonly used with FMCW signals.

`[resp,rnggrid] = step(response,x,xref)` computes the range response of the input signal, `x` using the reference signal, `xref`. This syntax applies when you set `RangeMethod` to `'FFT'` and `DechirpInput` to `true`. Often, the reference signal is the transmitted signal. This syntax assumes that the input signal has not been dechirped. This syntax is most commonly used with FMCW signals.

`[resp,rnggrid] = step(response,x,coeff)` computes the range response of `x` using the matched filter `coeff`. This syntax applies when you set `RangeMethod` to `'Matched filter'`. This syntax is most commonly used with pulsed signals.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`response` — Range response

`phased.RangeResponse` System object

Range response, specified as a `phased.RangeResponse` System object.

Example: `phased.RangeResponse`

x — Input radar data cube
complex-valued K-element column vector | complex-valued K-by-L matrix | complex-valued K-by-N-by-L array

Input radar data cube, specified as a complex-valued K-by-1 column vector, a K-by-L matrix, or K-by-N-by-L array.

- K is the number of fast-time or range samples.
- N is the number of independent spatial channels such as sensors or directions.
- L is the slow-time dimension that corresponds to the number of pulses or sweeps in the input signal.

See “Radar Data Cube”.

Each K-element fast-time dimension is processed independently.

For FMCW waveforms with a triangle sweep, the sweeps alternate between positive and negative slopes. However, `phased.RangeResponse` is designed to process consecutive sweeps of the same slope. To apply `phased.RangeResponse` for a triangle-sweep system, use one of the following approaches:

- Specify a positive `SweepSlope` property value, with `x` corresponding to upsweeps only. After obtaining the Doppler or speed values, divide them by 2.
- Specify a negative `SweepSlope` property value, with `x` corresponding to downsweeps only. After obtaining the Doppler or speed values, divide them by 2.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`

`xref` — Reference signal used for dechirping
complex-valued K-by-1 column vector

Reference signal used for dechirping, specified as a complex-valued K-by-1 column vector. The number of rows must equal the length of the fast-time dimension of `x`.

Dependencies

To enable this input argument, set the value of `RangeMethod` to `'FFT'` and `DechirpInput` to `true`.

Data Types: `single` | `double`

`coeff` — Matched filter coefficients
complex-valued P-by-1 column vector

Matched filter coefficients, specified as a complex-valued P-by-1 column vector. P must be less than or equal to K. K is the number of fast-time or range sample.

Dependencies

To enable this input argument, set the value of `RangeMethod` to `'Matched filter'`.

Data Types: `double`

Output Arguments

`resp` — Range response data cube
complex-valued *M*-element column vector | complex-valued *M*-by-*L* matrix | complex-valued *M*-by-*N* by-*L* array

Range response data cube, returned as one of the following:

- Complex-valued *M*-element column vector
- Complex-valued *M*-by-*L* matrix
- Complex-valued *M*-by-*N* by-*L* array

The value of *M* depends on the type of processing

RangeMethod Property	DechirpInput Property	Value of *M*
`'FFT'`	`false`	If you set the `RangeFFTLength` property to `'Auto'`, $M = K$, the length of the fast-time dimension of `x`. Otherwise, *M* equals the value of the `RangeFFTLength` property.
	`true`	*M* equals the quotient of the number of rows, *K*, of the input signal by the value of the decimation factor, *D*, specified in `DecimationFactor`.
`'Matched filter'`	n/a	$M = K$, the length of the fast-time dimension of `x`.

Data Types: `single` | `double`

`rnggrid` — Range values along range dimension
real-valued *M*-by-1 column vector

Range values along range dimension, returned as a real-valued *M*-by-1 column vector. `rnggrid` defines the ranges corresponding to the fast-time dimension of the `resp` output data cube. *M* is the length of the fast-time dimension of `resp`. Range values are monotonically increasing and equally spaced. Units are in meters.

Example: `[0,0.1,0.2,0.3]`

Data Types: `single` | `double`

Examples

Range Response of Three Targets

Compute the radar range response of three targets by using the `phased.RangeResponse` System object™. The transmitter and receiver are collocated isotropic antenna elements forming a monostatic radar system. The transmitted signal is a linear FM waveform with a pulse repetition interval of 7.0 μs and a duty cycle of 2%. The operating frequency is 77 GHz and the sample rate is 150 MHz.

```
fs = 150e6;
c = physconst('LightSpeed');
fc = 77e9;
pri = 7e-6;
prf = 1/pri;
```

Set up the scenario parameters. The radar transmitter and receiver are stationary and located at the origin. The targets are 500, 530, and 750 meters from the radars on the *x*-axis. The targets move along the *x*-axis at speeds of −60, 20, and 40 m/s. All three targets have a nonfluctuating radar cross-section (RCS) of 10 dB.

Create the target and radar platforms.

```
Numtgts = 3;
tgtpos = zeros(Numtgts);
tgtpos(1,:) = [500 530 750];
tgtvel = zeros(3,Numtgts);
tgtvel(1,:) = [-60 20 40];
tgtrcs = db2pow(10)*[1 1 1];
tgtmotion = phased.Platform(tgtpos,tgtvel);
target = phased.RadarTarget('PropagationSpeed',c,'OperatingFrequency',fc, ...
    'MeanRCS',tgtrcs);
radarpos = [0;0;0];
radarvel = [0;0;0];
radarmotion = phased.Platform(radarpos,radarvel);
```

Create the transmitter and receiver antennas.

```
txantenna = phased.IsotropicAntennaElement;
rxantenna = clone(txantenna);
```

Set up the transmitter-end signal processing. Create an upsweep linear FM signal with a bandwidth of half the sample rate. Find the length of the pri in samples and then estimate the rms bandwidth and range resolution.

```
bw = fs/2;
waveform = phased.LinearFMWaveform('SampleRate',fs, ...
    'PRF',prf,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',fs/2, ...
    'DurationSpecification','Duty cycle','DutyCycle',0.02);
sig = waveform();
Nr = length(sig);
bwrms = bandwidth(waveform)/sqrt(12);
rngrms = c/bwrms;
```

Set up the transmitter and radiator System object properties. The peak output power is 10 W and the transmitter gain is 36 dB.

```
peakpower = 10;
txgain = 36.0;
transmitter = phased.Transmitter( ...
    'PeakPower',peakpower, ...
    'Gain',txgain, ...
    'InUseOutputPort',true);
radiator = phased.Radiator( ...
    'Sensor',txantenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc);
```

Create a free-space propagation channel in two-way propagation mode.

```
channel = phased.FreeSpace( ...
    'SampleRate',fs, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc, ...
    'TwoWayPropagation',true);
```

Set up the receiver-end processing. The receiver gain is 42 dB and noise figure is 10.

```
collector = phased.Collector( ...
    'Sensor',rxantenna, ...
    'PropagationSpeed',c, ...
    'OperatingFrequency',fc);
rxgain = 42.0;
noisefig = 10;
receiver = phased.ReceiverPreamp( ...
    'SampleRate',fs, ...
    'Gain',rxgain, ...
    'NoiseFigure',noisefig);
```

Loop over 128 pulses to build a data cube. For each step of the loop, move the target and propagate the signal. Then put the received signal into the data cube. The data cube contains the received signal per pulse. Ordinarily, a data cube has three dimensions, where last dimension corresponds to antennas or beams. Because only one sensor is used in this example, the cube has only two dimensions.

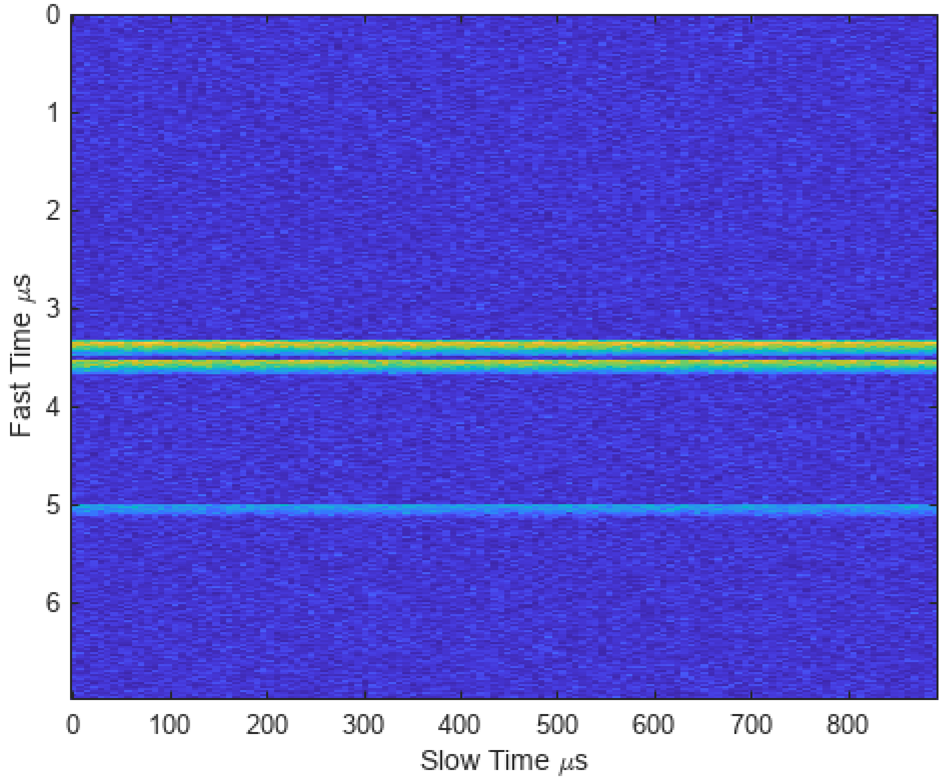
The processing steps are:

1. Move the radar and targets.
2. Transmit a waveform.
3. Propagate the waveform signal to the target.
4. Reflect the signal from the target.
5. Propagate the waveform back to the radar. Two-way propagation mode enables you to combine the returned propagation with the outbound propagation.
6. Receive the signal at the radar.
7. Load the signal into the data cube.

```
Np = 128;
cube = zeros(Nr,Np);
for n = 1:Np
    [sensorpos,sensorvel] = radarmotion(pri);
    [tgtpos,tgtvel] = tgtmotion(pri);
    [tgtrng,tgtang] = rangeangle(tgtpos,sensorpos);
    sig = waveform();
    [txsig,txstatus] = transmitter(sig);
    txsig = radiator(txsig,tgtang);
    txsig = channel(txsig,sensorpos,tgtpos,sensorvel,tgtvel);
    tgtsig = target(txsig);
    rxcol = collector(tgtsig,tgtang);
    rxsig = receiver(rxcol);
    cube(:,n) = rxsig;
end
```

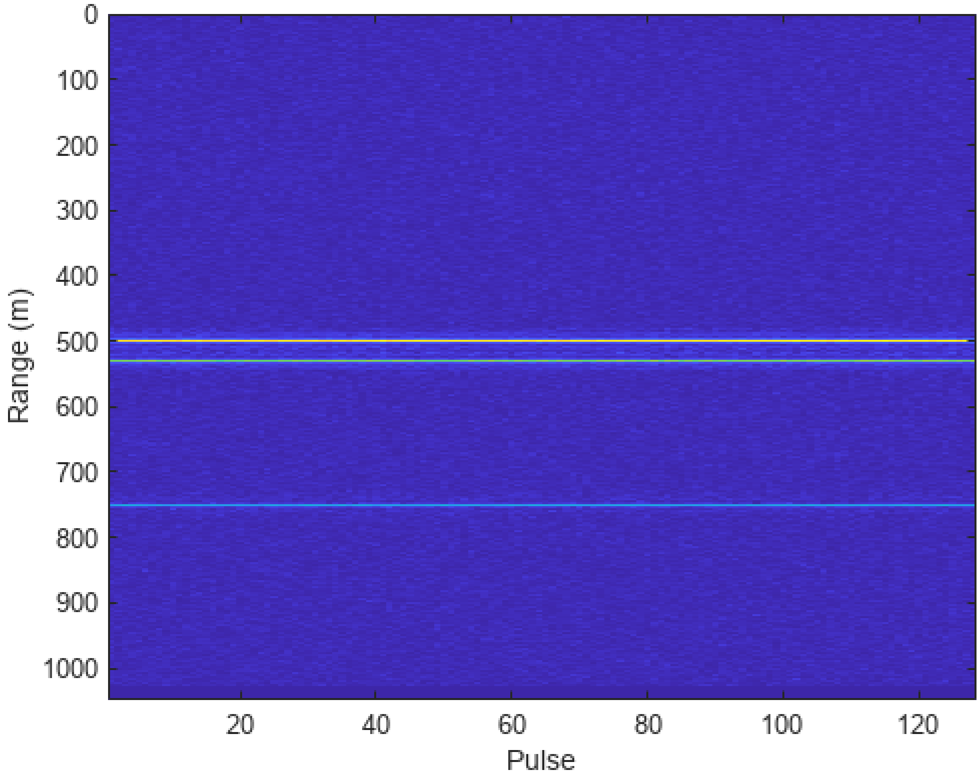
Display the image of the data cube containing signals per pulse.

```
imagesc([0:(Np-1)]*pri*1e6,[0:(Nr-1)]/fs*1e6,abs(cube))
xlabel('Slow Time {\mu}s')
ylabel('Fast Time {\mu}s')
```



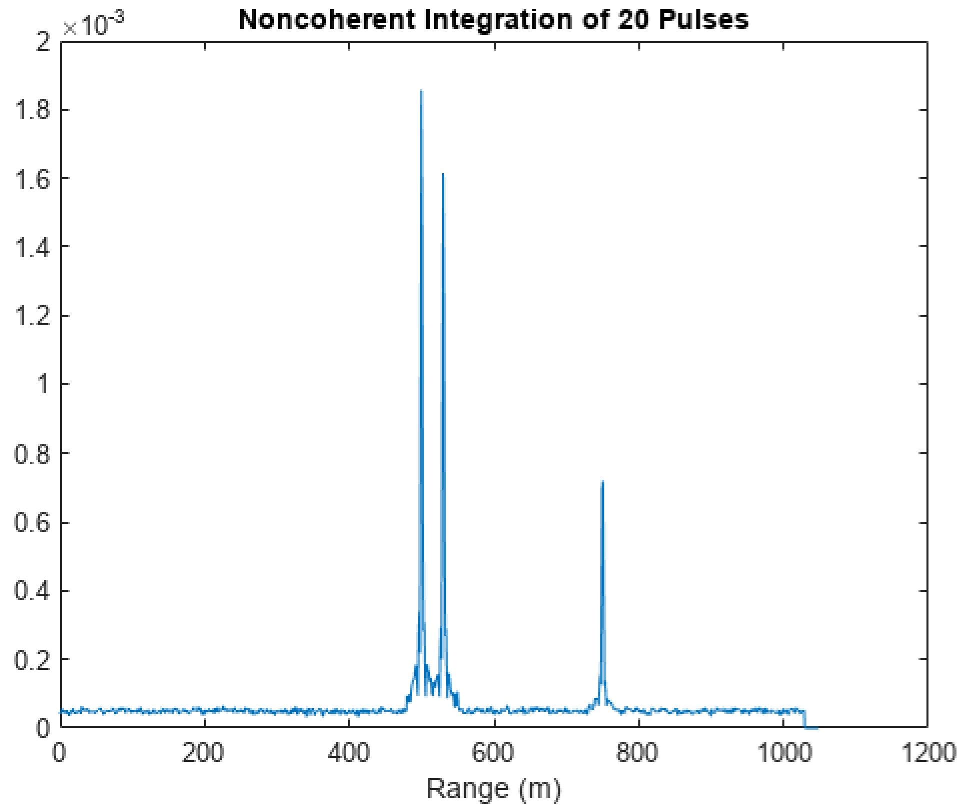
Create a phased.RangeResponse System object in matched filter mode. Then, display the range response image for the 128 pulses. The image shows range vertically and pulse number horizontally.

```
matchingcoeff = getMatchedFilter(waveform);
ndop = 128;
rangeresp = phased.RangeResponse('SampleRate',fs,'PropagationSpeed',c);
[resp,rnggrid] = rangeresp(cube,matchingcoeff);
imagesc([1:Np],rnggrid,abs(resp))
xlabel('Pulse')
ylabel('Range (m)')
```



Integrate 20 pulses noncoherently.

```
intpulse = pulsint(resp(:,1:20),'noncoherent');
plot(rnggrid,abs(intpulse))
xlabel('Range (m)')
title('Noncoherent Integration of 20 Pulses')
```



Version History

Introduced in R2017a

phased.ReceiverPreamp

Package: `phased`

Receiver preamp

Description

The `ReceiverPreamp` System object implements a model of a receiver preamplifier. The object receives incoming signals, multiplies them by the amplifier gain and divides by system losses. Finally, Gaussian white noise is added to the signal.

To model a receiver preamp:

1. Define and set up your receiver preamp. See "Construction" on page 1-1396.
2. Call `step` to amplify the input signal according to the properties of `phased.ReceiverPreamp`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.ReceiverPreamp` creates a receiver preamp System object, `H`.

`H = phased.ReceiverPreamp(Name,Value)` creates a receiver preamp object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`Gain`

Gain of receiver

A scalar containing the gain (in decibels) of the receiver preamp.

Default: `20`

`LossFactor`

Loss factor of receiver

A scalar containing the loss factor (in decibels) of the receiver preamp.

Default: `0`

`NoiseMethod`

Noise specification method

Specify how to compute noise power using one of `'Noise power'` | `'Noise temperature'`. If you set this property to `'Noise temperature'`, complex baseband noise is added to the input signal with noise power computed from the `ReferenceTemperature`, `NoiseFigure`, and `SampleRate` properties. If you set this property to `'Noise power'`, noise is added to the signal with power specified in the `NoisePower` property.

Default: `'Noise temperature'`

`NoiseFigure`

Noise figure of receiver

A scalar containing the noise figure (in decibels) of the receiver preamp. If the receiver has multiple channels/sensors, the noise figure applies to each channel/sensor. This property is only applicable when you set the `NoiseMethod` property to `'Noise temperature'`.

Default: `0`

`ReferenceTemperature`

Reference temperature of receiver

A scalar containing the reference temperature of the receiver (in kelvin). If the receiver has multiple channels/sensors, the reference temperature applies to each channel/sensor. This property is only applicable when you set the `NoiseMethod` property to `'Noise temperature'`.

Default: `290`

`SampleRate`

Sample rate

Specify the sample rate, in hertz, as a positive scalar. This property is only applicable when you set the `NoiseMethod` property to `'Noise temperature'`. The `SampleRate` property also specifies the noise bandwidth.

Default: `1e6`

`NoisePower`

Noise power

Specify the noise power (in Watts) as a positive scalar. This property is only applicable when you set the `NoiseMethod` property to `'Noise power'`.

Default: `1.0`

`NoiseComplexity`

Noise complexity

Specify the noise complexity as one of `'Complex'` | `'Real'`. When you set this property to `'Complex'`, the noise power is evenly divided between real and imaginary channels. Usually, complex-valued baseband signals require the addition of complex-valued noise. On occasion, when the signal is real-valued, you can use this option to specify that the noise is real-valued as well.

Default: `'Complex'`

`EnableInputPort`

Add input to specify enabling signal

To specify a receiver enabling signal, set this property to `true` and use the corresponding input argument when you invoke `step`. If you do not want to specify a receiver enabling signal, set this property to `false`.

Default: `false`

`PhaseNoiseInputPort`

Add input to specify phase noise

To specify the phase noise for each incoming sample, set this property to `true` and use the corresponding input argument when you invoke `step`. You can use this information to emulate coherent-on-receive systems. If you do not want to specify phase noise, set this property to `false`.

Default: `false`

`SeedSource`

Source of seed for random number generator

Specify how the object generates random numbers. Values of this property are:

`'Auto'`	The default MATLAB random number generator produces the random numbers. Use `'Auto'` if you are using this object with Parallel Computing Toolbox software.
`'Property'`	The object uses its own private random number generator to produce random numbers. The `Seed` property of this object specifies the seed of the random number generator. Use `'Property'` if you want repeatable results and are not using this object with Parallel Computing Toolbox software.

Default: `'Auto'`

`Seed`

Seed for random number generator

Specify the seed for the random number generator as a scalar integer between 0 and $2^{32}–1$. This property applies when you set the `SeedSource` property to `'Property'`.

Default: `0`

Methods

reset	Reset random number generator for noise generation
step	Receive incoming signal

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Preamplify Signal

This example shows how to use the `phased.ReceiverPreamp` System object™ to amplify a sine wave.

Create a `phased.ReceiverPreamp` System object with a sample rate of 100 Hz. Assume a receiver noise figure of 60 dB.

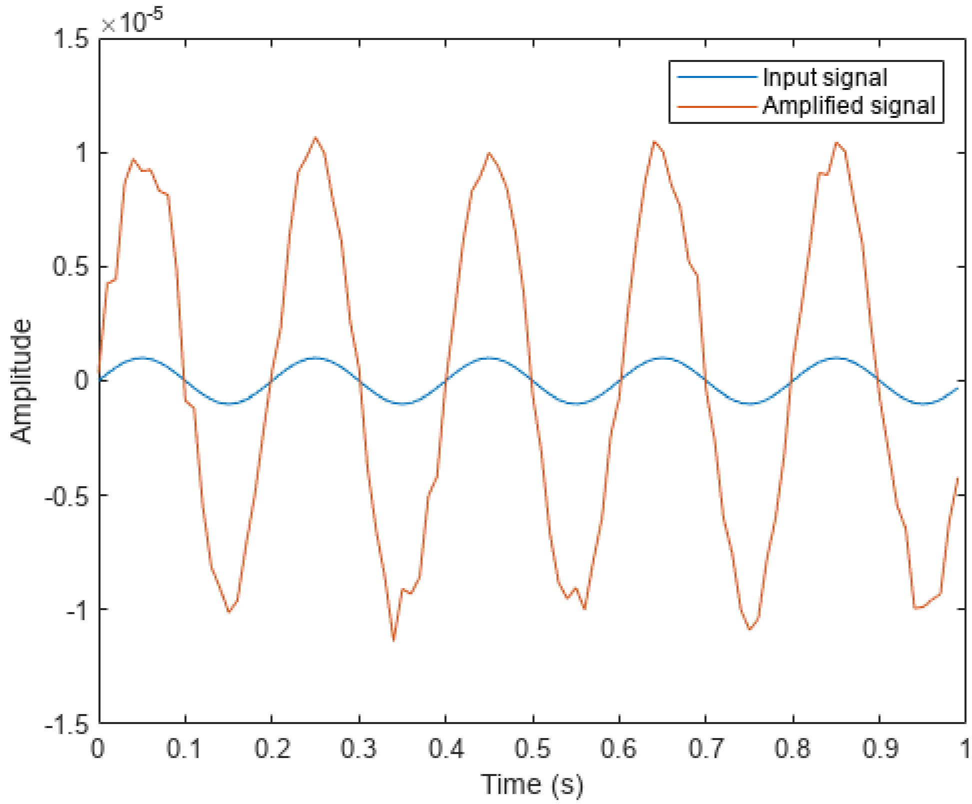
```
fs = 100;
receiver = phased.ReceiverPreamp('NoiseFigure',60, ...
    'SampleRate',fs,'NoiseComplexity','Real');
```

Create the input signal.

```
t = linspace(0,1-1/fs,100);
x = 1e-6*sin(2*pi*5*t);
```

Amplify the signal and compare it with the input signal.

```
y = receiver(x);
plot(t,x,t,real(y))
xlabel('Time (s)')
ylabel('Amplitude')
legend('Input signal','Amplified signal')
```



Version History

Introduced in R2011a

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.Collector` | `phased.Transmitter`

Topics

"Receiver Preamp"

reset

System object: `phased.ReceiverPreamp`
Package: `phased`

Reset random number generator for noise generation

Syntax

`reset(H)`

Description

`reset(H)` resets the states of the `ReceiverPreamp` object, `H`. This method resets the random number generator state if the `SeedSource` property is set to `'Property'`.

step

System object: `phased.ReceiverPreamp`
Package: `phased`

Receive incoming signal

Syntax

```
Y = step(H,X)
Y = step(H,X,EN_RX)
Y = step(H,X,PHNOISE)
Y = step(H,X,EN_RX,PHNOISE)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` applies the receiver gain and the receiver noise to the input signal, `X`, and returns the resulting output signal, `Y`.

`Y = step(H,X,EN_RX)` uses input `EN_RX` as the enabling signal when the `EnableInputPort` property is set to `true`.

`Y = step(H,X,PHNOISE)` uses input `PHNOISE` as the phase noise for each sample in `X` when the `PhaseNoiseInputPort` is set to `true`. The phase noise is the same for all channels in `X`. The elements in `PHNOISE` represent the random phases the transmitter adds to the transmitted pulses. The receiver preamp object removes these random phases from all received samples returned within corresponding pulse intervals. Such setup is often referred to as *coherent on receive*.

`Y = step(H,X,EN_RX,PHNOISE)` combines all input arguments. This syntax is available when you configure `H` so that `H.EnableInputPort` is `true` and `H.PhaseNoiseInputPort` is `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Receiver object.

X

Input signal

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

EN_RX

Enabling signal, specified as a column vector whose length equals the number of rows in X. The data type of EN_RN is double or logical. Every element of EN_RX that equals 0 or false indicates that the receiver is turned off, and no input signal passes through the receiver. Every element of EN_RX that is nonzero or true indicates that the receiver is turned on, and the input passes through.

PHNOISE

Phase noise for each sample in X, specified as a column vector whose length equals the number of rows in X. You can obtain PHNOISE as an optional output argument from the step method of phased.Transmitter.

Output Arguments

Y

Output signal. Y has the same dimensions as X.

Examples

Preamplify Cosine Signal

This example shows how to construct a phased.ReceiverPreamp System object™ with a noise figure of 5 dB and a bandwidth of 1 MHz. Then use the object to amplify the signal.

Construct the Receiver Preamp System object.

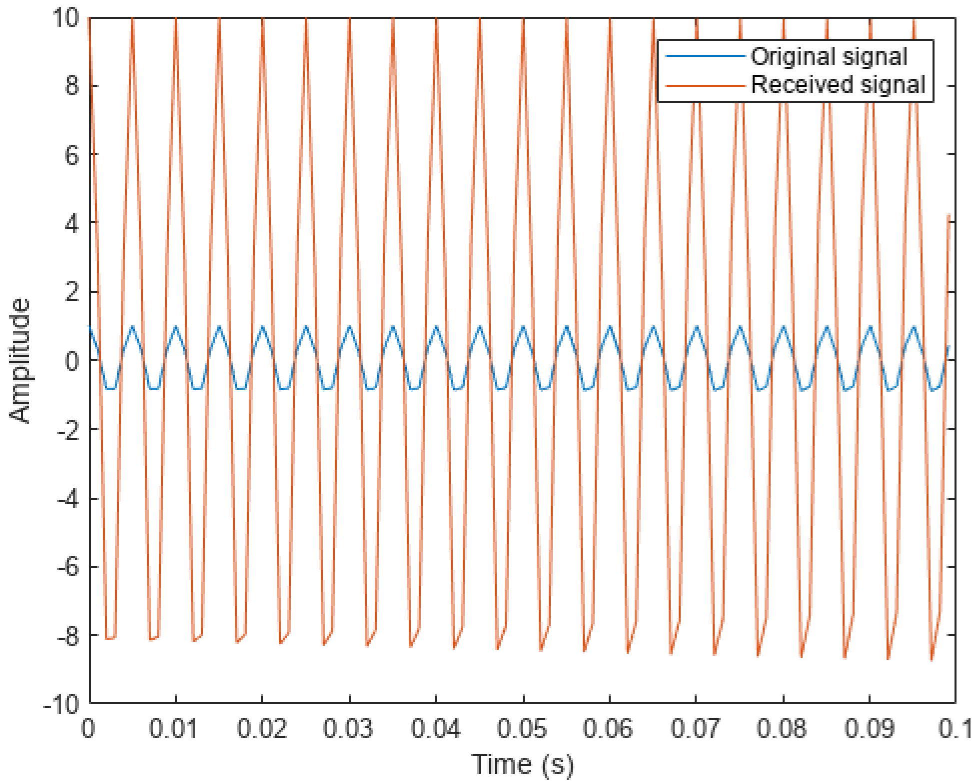
```
receiver = phased.ReceiverPreamp('NoiseFigure',5,'SampleRate',1e6);
```

Create the signal.

```
Fs = 1e3;
t = linspace(0,1,1e3);
x = cos(2*pi*200*t)';
```

Use the step method to amplify the signal and then plot the first 100 samples.

```
y = receiver(x);
idx = [1:100];
plot(t(idx),x(idx),t(idx),real(y(idx)))
xlabel('Time (s)')
ylabel('Amplitude')
legend('Original signal','Received signal')
```



Original signal
Received signal
Amplitude
Time (s)
10
8
6
4
2
0
-2
-4
-6
-8
-10
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1

phased.RectangularWaveform

Package: phased

Rectangular pulse waveform

Description

The RectangularWaveform object creates a rectangular pulse waveform.

To obtain waveform samples:

1. Define and set up your rectangular pulse waveform. See "Construction" on page 1-1406.
2. Call step to generate the rectangular pulse waveform samples according to the properties of phased.RectangularWaveform. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations. When the only argument to the step method is the System object itself, replace y = step(obj) by y = obj().

Construction

H = phased.RectangularWaveform creates a rectangular pulse waveform System object, H. The object generates samples of a rectangular pulse.

H = phased.RectangularWaveform(Name,Value) creates a rectangular pulse waveform object, H, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

SampleRate

Sample rate

Signal sample rate, specified as a positive scalar. Units are Hertz. The ratio of sample rate to pulse repetition frequency *(PRF)* must be a positive integer — each pulse must contain an integer number of samples.

Default: 1e6

DurationSpecification

Method to set pulse duration

Method to set pulse duration (pulse width), specified as 'Pulse width' or 'Duty cycle'. This property determines how you set the pulse duration. When you set this property to 'Pulse width', then you set the pulse duration directly using the PulseWidth property. When you set this property to `'Duty cycle'`, you set the pulse duration from the values of the `PRF` and `DutyCycle` properties. The pulse width is equal to the duty cycle divided by the *PRF*.

Default: `'Pulse width'`

`PulseWidth`

Pulse width

Specify the length of each pulse (in seconds) as a positive scalar. The value must satisfy `PulseWidth <= 1./PRF`.

Default: `50e-6`

`DutyCycle`

Waveform duty cycle

Waveform duty cycle, specified as a scalar from 0 to 1, exclusive. This property applies when you set the `DurationSpecification` property to `'Duty cycle'`. The pulse width is the value of the `DutyCycle` property divided by the value of the `PRF` property.

Default: `0.5`

`PRF`

Pulse repetition frequency

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The*PRF* must satisfy these restrictions:

- The product of *PRF* and *PulseWidth* must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of `PRF` must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* using property settings alone or using property settings in conjunction with the `prfidx` input argument of the `step` method.

- When `PRFSelectionInputPort` is `false`, you set the *PRF* using properties only. You can
  - implement a constant *PRF* by specifying `PRF` as a positive real-valued scalar.
  - implement a staggered *PRF* by specifying `PRF` as a row vector with positive real-valued entries. Then, each call to the `step` method uses successive elements of this vector for the *PRF*. If the last element of the vector is reached, the process continues cyclically with the first element of the vector.
- When `PRFSelectionInputPort` is `true`, you can implement a selectable *PRF* by specifying `PRF` as a row vector with positive real-valued entries. But this time, when you execute the `step` method, select a *PRF* by passing an argument specifying an index into the *PRF* vector.

In all cases, the number of output samples is fixed when you set the `OutputFormat` property to `'Samples'`. When you use a varying *PRF* and set the `OutputFormat` property to `'Pulses'`, the number of samples can vary.

Default: 10e3

PRFSelectionInputPort

Enable PRF selection input

Enable the PRF selection input, specified as true or false. When you set this property to false, the step method uses the values set in the PRF property. When you set this property to true, you pass an index argument into the step method to select a value from the PRF vector.

Default: false

FrequencyOffsetSource

Source of frequency offset

Source of frequency offset for the waveform, specified as 'Property' or 'Input port'.

- When you set this property to 'Property', the offset is determined by the value of the FrequencyOffset property.
- When you set this property to 'Input port', the FrequencyOffset is determined by the freqoffset input argument.

Default: 'Property'

FrequencyOffset

Frequency offset

Frequency offset in Hz, specified as a scalar.

Dependencies

This property applies when you set the FrequencyOffsetSource property to 'Input port'.

Default: 0 Hz

OutputFormat

Output signal format

Specify the format of the output signal as 'Pulses' or 'Samples'. When you set the OutputFormat property to 'Pulses', the output of the step method takes the form of multiple pulses specified by the value of the NumPulses property. The number of samples per pulse can vary if you change the pulse repetition frequency during the simulation.

When you set the OutputFormat property to 'Samples', the output of the step method is in the form of multiple samples. In this case, the number of output signal samples is the value of the NumSamples property and is fixed.

Default: 'Pulses'

NumSamples

Number of samples in output

Specify the number of samples in the output of the `step` method as a positive integer. This property applies only when you set the `OutputFormat` property to `'Samples'`.

Default: `100`

`NumPulses`

Number of pulses in output

Specify the number of pulses in the output of the `step` method as a positive integer. This property applies only when you set the `OutputFormat` property to `'Pulses'`.

Default: `1`

`PRFOutputPort`

Set this property to `true` to output the PRF for the current pulse using a `step` method argument.

Dependencies

This property can be used only when the `OutputFormat` property is set to `'Pulses'`.

Default: `false`

`CoefficientsOutputPort`

Enable matched filter coefficients output port

Enable the matched filter coefficients output port, specified as `false` or `true`. When you set this property to `false`, the object does not provide the matched filter coefficients used during the simulation as an output. When you set this property to `true`, the object provides the matched filter coefficients used during the simulation as an output.

Default: `false`

Methods

bandwidth	Bandwidth of rectangular pulse waveform
getMatchedFilter	Matched filter coefficients for waveform
plot	Plot rectangular pulse waveform
reset	Reset states of rectangular waveform object
step	Samples of rectangular pulse waveform

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Plot Rectangular Waveform and Spectrum

Create and plot a rectangular pulse waveform object and then plot its spectrum.

Plot the waveform

Create and plot a pulse waveform. The sample rate is 500 kHz, the pulse width is 0.1 millisecond. The pulse repetition interval is twice the pulse duration.

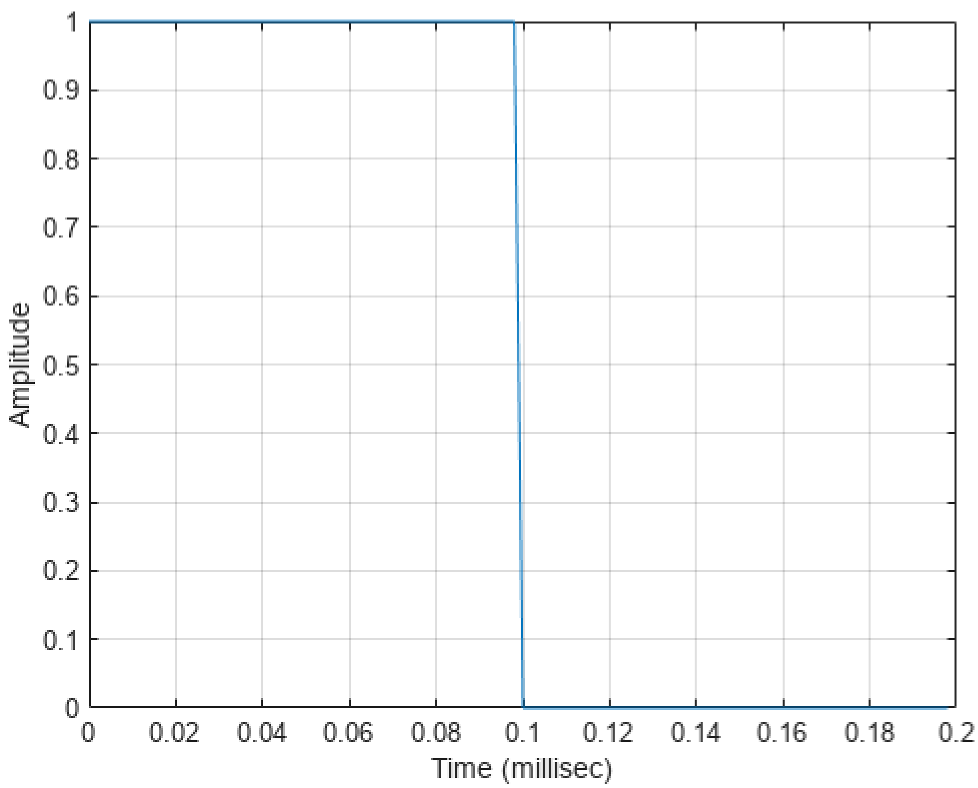
```
fs = 500e3;
```

Create the rectangular waveform System object™.

```
sWF = phased.RectangularWaveform('SampleRate',fs,'PulseWidth',1e-4,'PRF',5000.0);
```

Use the step method to obtain the waveform. Then, plot the waveform.

```
rectwav = step(sWF);
nsamp = size(rectwav,1);
t = [0:(nsamp-1)]/fs;
plot(t*1000,real(rectwav))
xlabel('Time (millisec)')
ylabel('Amplitude')
grid
```

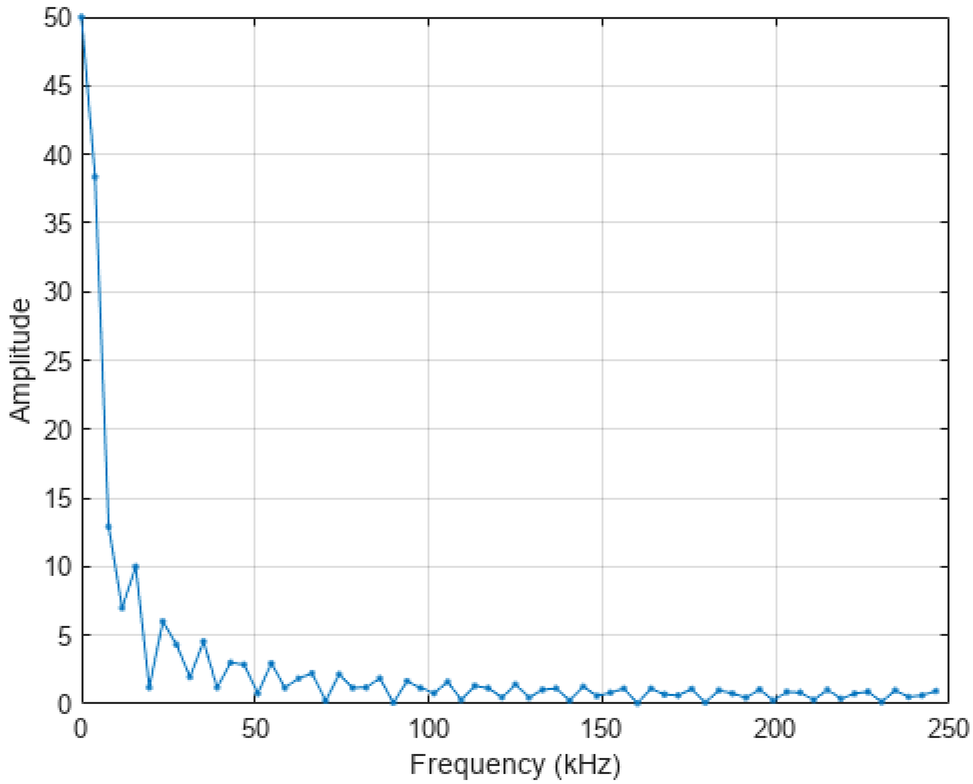


Plot the spectrum

Compute the Fourier transform of the complex signal. Then show the spectrum.

```
nfft = 2^nextpow2(nsamp);
Z = fft(real(rectwav),nfft);
fr = [0:(nfft/2-1)]/nfft*fs;
```

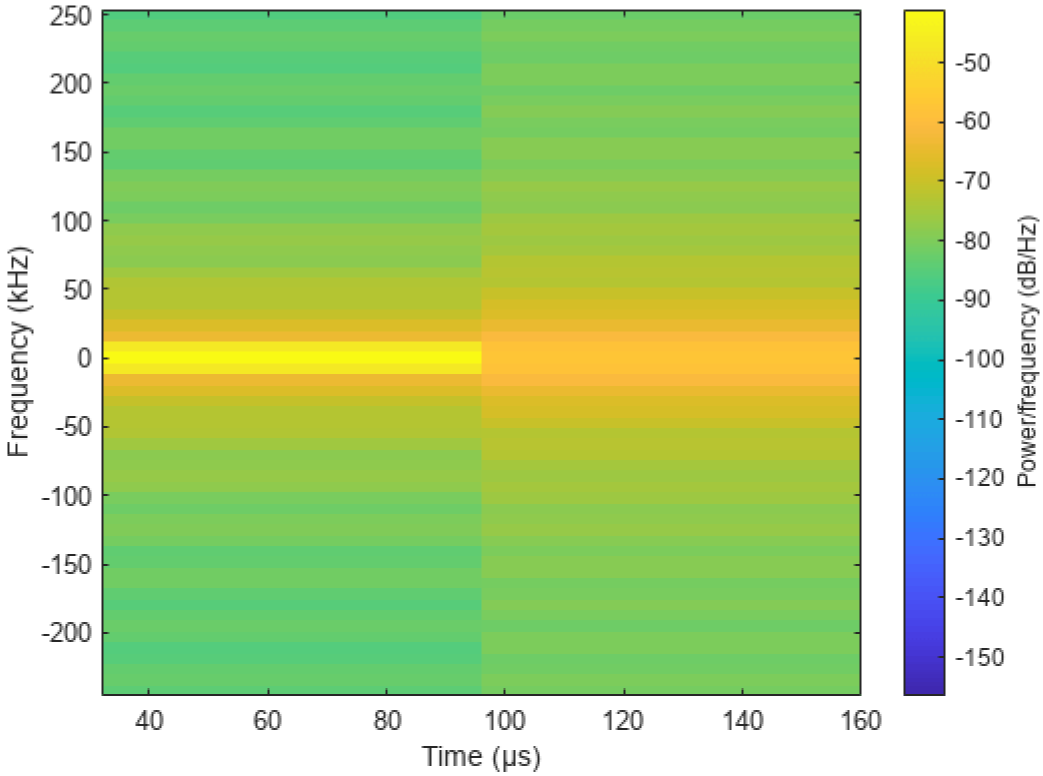
```
plot(fr/1000,abs(Z(1:nfft/2)),'.-')
xlabel('Frequency (kHz)')
ylabel('Amplitude')
grid
```



Plot the spectrogram

Plot a spectrogram of the function with a window size of 64 samples and 50% overlap. Window the signal with a Hamming function.

```
nfft1 = 64;
nov = floor(0.5*nfft1);
spectrogram(rectwav,hamming(nfft1),nov,nfft1,fs,'centered','yaxis')
```



This plot shows the constant frequency of the signal.

Apply Frequency Offset to Rectangular Waveform

Apply a frequency offset to a rectangular pulse waveform. Plot the frequency spectrum of the waveform with and without a frequency offset applied.

Create a rectangular waveform object which is configured to set the frequency offset from an input when the object is executed.

```
fs = 500e3;
sRWF = phased.RectangularWaveform('SampleRate',fs,'PulseWidth',1e-4, ...
    'PRF',5000.0,'FrequencyOffsetSource','Input port');
```

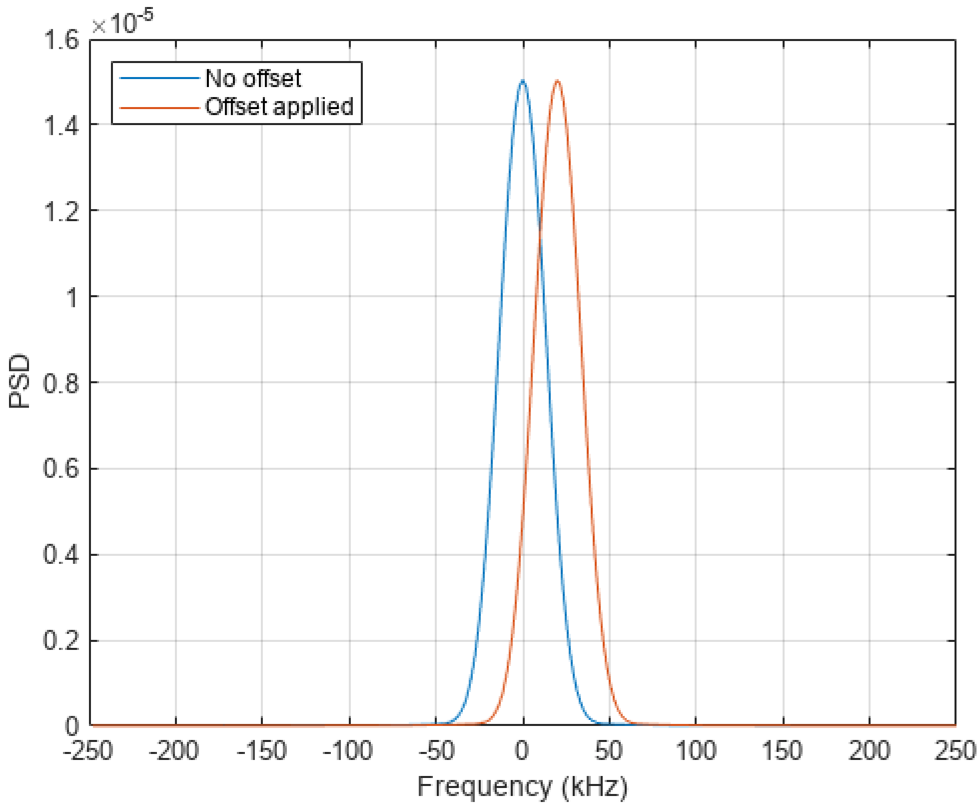
Execute the object two times. First set the frequency offset set to 0 Hz, and then to 2e4 Hz.

```
rectwav = sRWF(0);
rectwav_foffset = sRWF(2e4);
```

Plot the frequency spectrum of the complex signals. The frequency offset signal is shifted to the right.

```
[Pxx,f] = pwelch(rectwav,[],[],[],fs,'centered');
[Pxx_offset,foffset] = pwelch(rectwav_foffset,[],[],[],fs,'centered');
plot(f/1000,Pxx,foffset/1000,Pxx_offset)
ylabel('PSD');
xlabel('Frequency (kHz)');
```

```
legend({'No offset','Offset applied'},'Location','northwest');
grid on;
```



Version History

Introduced in R2011a

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `plot` method is not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.LinearFMWaveform` | `phased.SteppedFMWaveform` | `phased.PhaseCodedWaveform`

Topics

Waveform Analysis Using the Ambiguity Function

bandwidth

System object: `phased.RectangularWaveform`
Package: `phased`

Bandwidth of rectangular pulse waveform

Syntax

```
BW = bandwidth(H)
```

Description

`BW = bandwidth(H)` returns the bandwidth (in hertz) of the pulses for the rectangular pulse waveform, `H`. The bandwidth equals the reciprocal of the pulse width.

Input Arguments

H

Rectangular pulse waveform object.

Output Arguments

BW

Bandwidth of the pulses, in hertz.

Examples

Find Bandwidth of Rectangular Pulse

Determine the bandwidth of a rectangular pulse waveform.

```
waveform = phased.RectangularWaveform;
bw = bandwidth(waveform)
```

```
bw = 20000
```

getMatchedFilter

System object: `phased.RectangularWaveform`
Package: `phased`

Matched filter coefficients for waveform

Syntax

```
Coeff = getMatchedFilter(H)
Coeff = getMatchedFilter(H,'FrequencyOffset',FOFFSET)
```

Description

`Coeff = getMatchedFilter(H)` returns the matched filter coefficients for the rectangular waveform object `H`. `Coeff` is a column vector.

`Coeff = getMatchedFilter(H,'FrequencyOffset',FOFFSET)` adds a frequency offset when matched filter coefficients are generated. `FOFFSET` must be a scalar. This option is available when you set the `FrequencyOffsetSource` property to `'Input port'` for the input object, `H`.

Examples

Matched Filter Coefficients for Rectangular Pulse

Get the matched filter coefficients for a rectangular pulse waveform.

```
waveform = phased.RectangularWaveform('PulseWidth',1e-5,...
    'OutputFormat','Pulses','NumPulses',1);
coeff = getMatchedFilter(waveform)
```

```
coeff = 10×1

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
```

plot

System object: phased.RectangularWaveform
Package: phased

Plot rectangular pulse waveform

Syntax

```
plot(Hwav)
plot(Hwav,Name,Value)
plot(Hwav,Name,Value,LineSpec)
h = plot( ___ )
```

Description

plot(Hwav) plots the real part of the waveform specified by Hwav.

plot(Hwav,Name,Value) plots the waveform with additional options specified by one or more Name,Value pair arguments.

plot(Hwav,Name,Value,LineSpec) specifies the same line color, line style, or marker options as are available in the MATLAB plot function.

h = plot(___) returns the line handle in the figure.

Input Arguments

Hwav

Waveform object. This variable must be a scalar that represents a single waveform object.

LineSpec

Character vector to specifies the same line color, style, or marker options as are available in the MATLAB plot function. If you specify a PlotType value of 'complex', then LineSpec applies to both the real and imaginary subplots.

Default: 'b'

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

PlotType

Specifies whether the function plots the real part, imaginary part, or both parts of the waveform. Valid values are 'real', 'imag', and 'complex'.

Default: 'real'

PulseIdx

Index of the pulse to plot. This value must be a scalar.

Default: 1

FrequencyOffset

Frequency offset

Frequency offset in Hz, specified as a scalar.

Dependencies

This property applies when you set the FrequencyOffsetSource property to 'Input port'.

Default: 0 Hz

Output Arguments

h

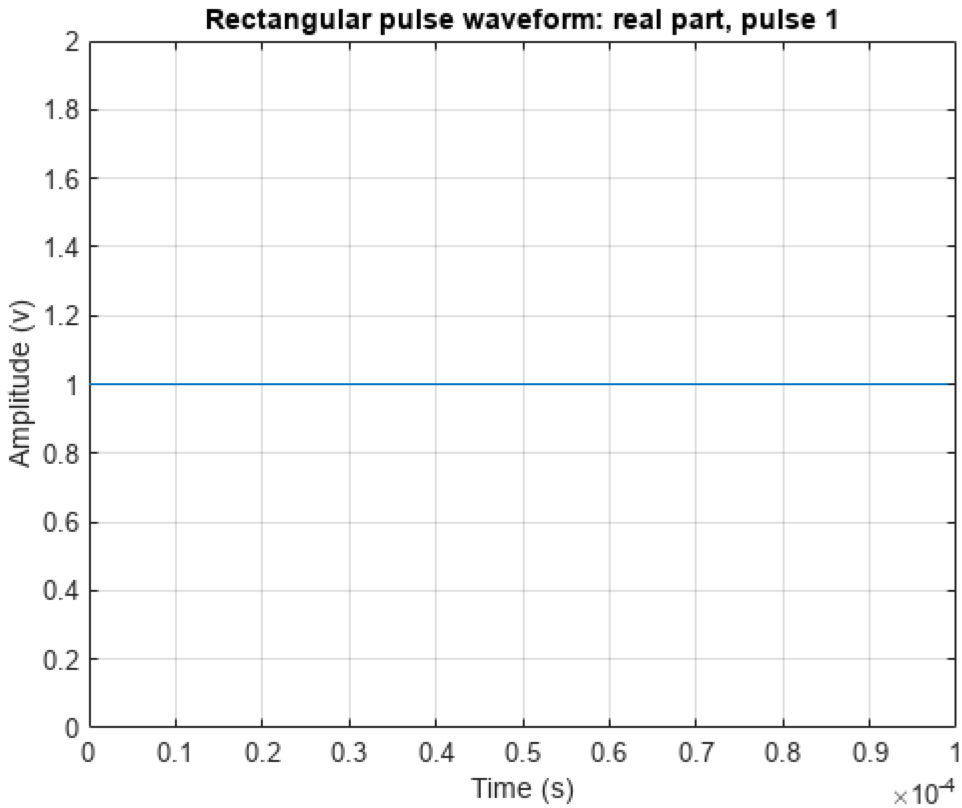
Handle to the line or lines in the figure. For a PlotType value of 'complex', h is a column vector. The first and second elements of this vector are the handles to the lines in the real and imaginary subplots, respectively.

Examples

Plot Rectangular Waveform

Create and plot a 100 µs rectangular pulse waveform.

```
waveform = phased.RectangularWaveform('PulseWidth',100e-6);
plot(waveform);
```



Rectangular pulse waveform: real part, pulse 1
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Amplitude (v)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Time (s)
×10^-4

reset

System object: `phased.RectangularWaveform`
Package: `phased`

Reset states of rectangular waveform object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `RectangularWaveform` object, `H`. Afterward, if the `PRF` property is a vector, the next call to `step` uses the first PRF value in the vector.

step

System object: `phased.RectangularWaveform`
Package: `phased`

Samples of rectangular pulse waveform

Syntax

```
Y = step(sRFM)
Y = step(sRFM,prfidx)
Y = step(sRFM,freqoffset)
[Y,PRF] = step( ___ )
[Y,COEFF] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

`Y = step(sRFM)` returns samples of a rectangular pulse in the column vector `Y`.

`Y = step(sRFM,prfidx)`, uses the `prfidx` index to select the PRF from the predefined vector of values specified by the `PRF` property. This syntax applies when you set the `PRFSelectionInputPort` property to `true`.

`Y = step(sRFM,freqoffset)`, uses the `freqoffset` to generate the waveform with an offset as specified at step time. Use this syntax for cases where the transmit pulse frequency needs to be dynamically updated. This syntax applies when you set the `FrequencyOffsetSource` property to `'Input port'`.

`[Y,PRF] = step( ___ )` also returns the current pulse repetition frequency, `PRF`. To enable this syntax, set the `PRFOutputPort` property to `true` and set the `OutputFormat` property to `'Pulses'`.

`[Y,COEFF] = step( ___ )` returns the matched filter coefficients, `COEFF`, for the current pulse. To enable this syntax, set `CoefficientsOutputPort` to `true`. `COEFF` is returned as either an N_Z-by-1 vector or an N_Z-by-M matrix.

- An N_Z-by-1 vector is returned when:
  - The object has `OutputFormat` set to `'Pulses'` and `NumPulses` is equal to 1. N_Z is the pulse width.
  - The object is configured to generate constant pulse width waveforms (`DurationSpecification` is set to `'Pulse width'` or `'Duty cycle'` and `PRF` has one unique value); and either `OutputFormat` is set to `'Pulses'` and `NumPulses` is greater than 1, or `OutputFormat` is set to `'Samples'`. For this case, N_Z is the pulse width.

- An N_Z-by-M matrix is returned when the object generates varying pulse widths (`DurationSpecification` is set to `'Duty cycle'` and `PRF` has more than one unique value); and either `OutputFormat` set to `'Pulses'` and `NumPulses` is greater than 1, or `OutputFormat` is set to `'Samples'`. For this case, N_Z is the pulse width, and M is the number of sub-pulses, `NumSteps`.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example, `[Y,PRF,COEFF] = step(sRFM,prfidx,freqoffset)`.

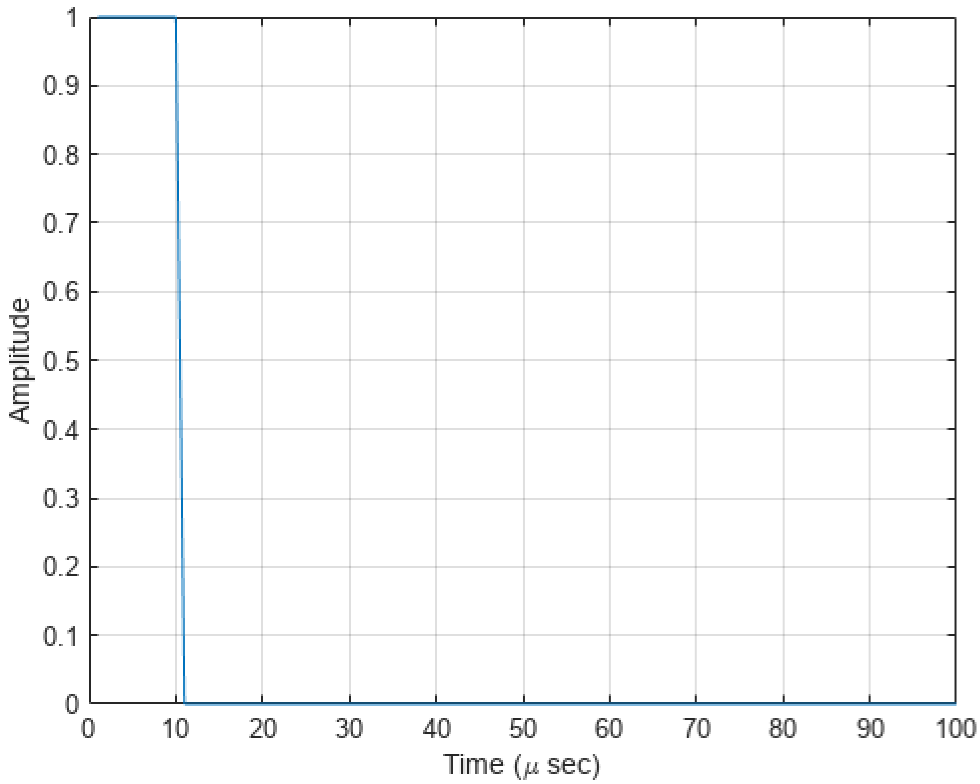
Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Create Rectangular Waveform Pulse

Construct a 10 microseconds rectangular pulse with a pulse repetition interval of 100 microseconds.

```
Pulsewidth = 10e-6;
PRI = 100e-6;
sRFM = phased.RectangularWaveform('PulseWidth',Pulsewidth,...
    'OutputFormat','Pulses','NumPulses',1,...
    'SampleRate',1e6,'PRF',1/PRI);
wav = step(sRFM);
plot(wav)
xlabel('Time (\mu sec)')
ylabel('Amplitude')
grid
```



Create Rectangular Pulses with Variable PRF

Construct rectangular waveforms with two pulses each. Set the sample rate to 1 MHz, a pulse width of 50 microseconds, and a duty cycle of 20%. Vary the pulse repetition frequency.

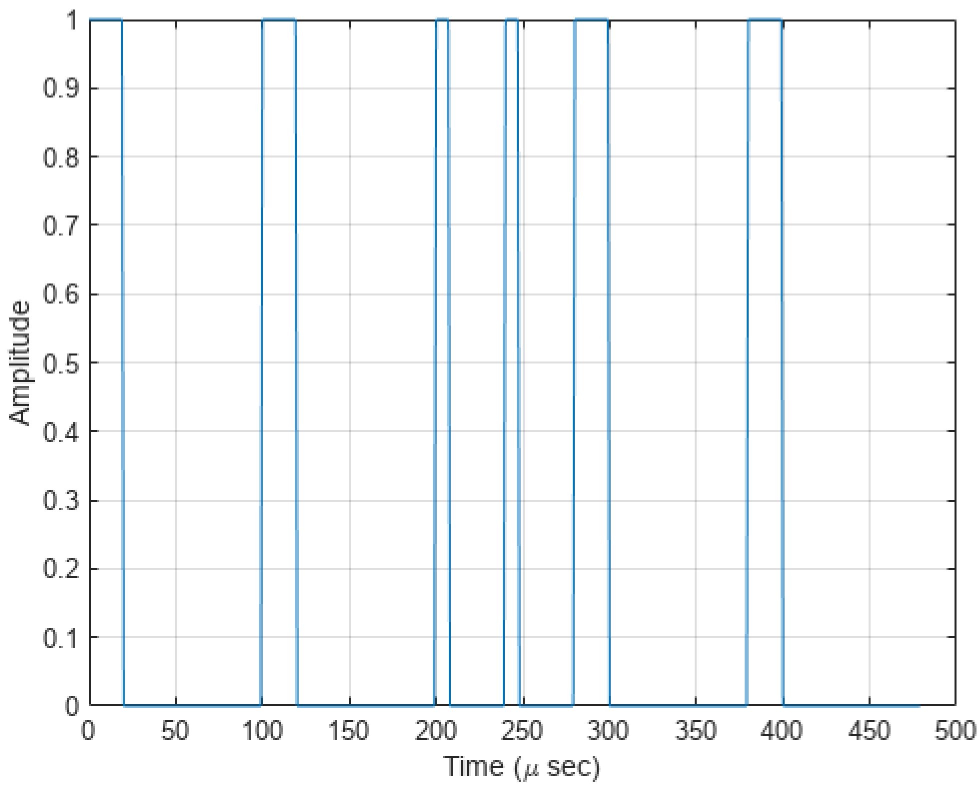
Set the sample rate and PRF. The ratio of sample rate to PRF must be an integer.

```
fs = 1e6;
PRF = [10000,25000];
waveform = phased.RectangularWaveform('OutputFormat','Pulses','SampleRate',fs,...
    'DurationSpecification','Duty Cycle','DutyCycle',.2,...
    'PRF',PRF,'NumPulses',2,'PRFSelectionInputPort',true);
```

Obtain and plot the rectangular waveforms. For the first call to the `step` method, set the PRF to 10 kHz using the PRF index. For the next call, set the PRF to 25 kHz. For the final call, set the PRF to 10 kHz.

```
wav = [];
wav1 = waveform(1);
wav = [wav; wav1];
wav1 = waveform(2);
wav = [wav; wav1];
wav1 = waveform(1);
wav = [wav; wav1];
nsamps = size(wav,1);
```

```
t = [0:(nsamps-1)]/waveform.SampleRate;
plot(t*1e6,real(wav))
xlabel('Time (\mu sec)')
ylabel('Amplitude')
grid
```



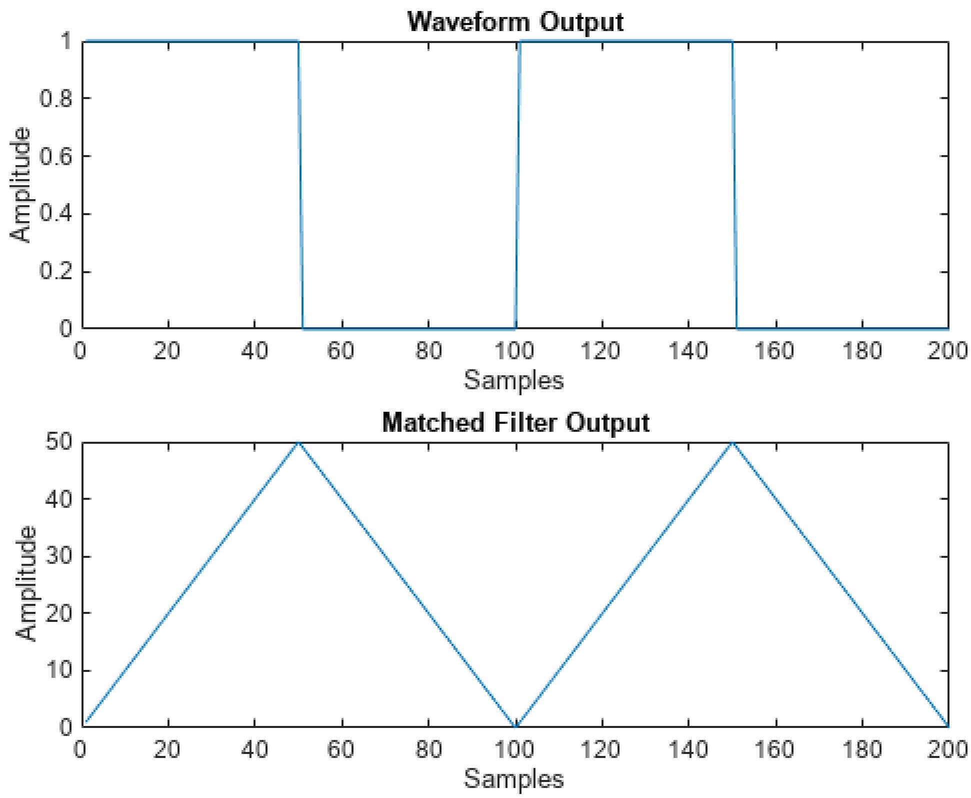
Generate Matched Filter Coefficients of Rectangular Pulse Waveform

Generate output samples and matched filter coefficients of a rectangular pulse waveform.

```
waveform = phased.RectangularWaveform('CoefficientsOutputPort',true, ...
    'PRF',[1e4 2e4],'DurationSpecification','Duty cycle','DutyCycle',0.5, ...
    'OutputFormat','Pulses','NumPulses',2,'PRFSelectionInputPort',true);
[wav,coeff] = waveform(1);
```

Create a matched filter that applies the coefficients as an input argument. Use the coefficients when applying the matched filter to the waveform. Plot the waveform and matched filter outputs.

```
mf = phased.MatchedFilter('CoefficientsSource','Input port');
mfOut = mf(wav,coeff(:,1));
subplot(211),plot(real(wav));
xlabel('Samples'),ylabel('Amplitude'),title('Waveform Output');
subplot(212),plot(abs(mfOut));
xlabel('Samples'),ylabel('Amplitude'),title('Matched Filter Output');
```


Waveform Output
Amplitude
Samples
1
0.8
0.6
0.4
0.2
0
0
20
40
60
80
100
120
140
160
180
200
Matched Filter Output
Amplitude
Samples
50
40
30
20
10
0

phased.ReplicatedSubarray

Package: `phased`

Phased array formed by replicated subarrays

Description

The `ReplicatedSubarray` object represents a phased array that contains copies of a subarray created by replicating a single specified array.

To obtain the response of the subarrays:

1. Define and set up your phased array containing replicated subarrays. See “Construction” on page 1-1426.
2. Call `step` to compute the response of the subarrays according to the properties of `phased.ReplicatedSubarray`. The behavior of `step` is specific to each object in the toolbox.

You can also use a `ReplicatedSubarray` object as the value of the `SensorArray` or `Sensor` property of objects that perform beamforming, steering, and other operations.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.ReplicatedSubarray` creates a replicated subarray System object, `H`. This object represents an array that contains copies of a subarray.

`H = phased.ReplicatedSubarray(Name,Value)` creates a replicated subarray object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`Subarray`

Subarray to replicate

Specify the subarray you use to form the array. The subarray must be a `phased.ULA`, `phased.URA`, or `phased.ConformalArray` object.

Default: `phased.ULA` with default property values

`Layout`

Layout of subarrays

Specify the layout of the replicated subarrays as `'Rectangular'` or `'Custom'`.

Default: `'Rectangular'`

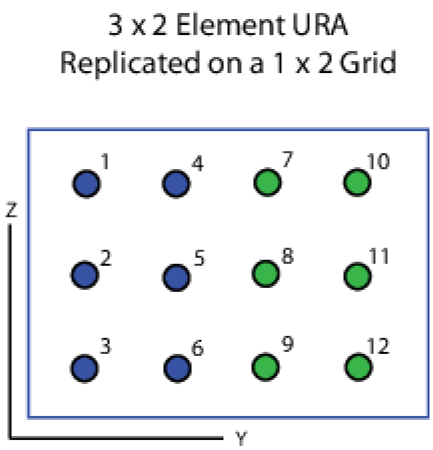
`GridSize`

Size of rectangular grid

Specify the size of the rectangular grid as a single positive integer or 1-by-2 positive integer row vector. This property applies only when you set the `Layout` property to `'Rectangular'`.

If `GridSize` is a scalar, the array has the same number of subarrays in each row and column.

If `GridSize` is a 1-by-2 vector, the vector has the form `[NumberOfRows, NumberOfColumns]`. The first entry is the number of subarrays along each column, while the second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. This figure shows how a 3-by-2 URA subarray is replicated using a `GridSize` value of `[1,2]`.



Default: `[1 2]`

`GridSpacing`

Spacing of rectangular grid

Specify the rectangular grid spacing of subarrays as a positive real-valued scalar, a 1-by-2 row vector, or `'Auto'`. This property applies only when you set the `Layout` property to `'Rectangular'`. Grid spacing units are expressed in meters.

If `GridSpacing` is a scalar, the spacing along the row and the spacing along the column is the same.

If `GridSpacing` is a length-2 row vector, it has the form `[SpacingBetweenRows, SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.

If `GridSpacing` is `'Auto'`, the replication preserves the element spacing in both row and column. This option is available only if you use a `phased.ULA` or `phased.URA` object as the subarray.

Default: `'Auto'`

`SubarrayPosition`

Subarray positions in custom grid

Specify the positions of the subarrays in the custom grid. This property value is a 3-by-N matrix, where N indicates the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array's local coordinate system, in meters, using the form [x; y; z].

This property applies when you set the `Layout` property to `'Custom'`.

Default: `[0 0; -0.5 0.5; 0 0]`

`SubarrayNormal`

Subarray normal directions in custom grid

Specify the normal directions of the subarrays in the array. This property value is a 2-by-N matrix, where N is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form [azimuth; elevation]. Each angle is in degrees and is defined in the local coordinate system.

You can use the `SubarrayPosition` and `SubarrayNormal` properties to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

This property applies when you set the `Layout` property to `'Custom'`.

Default: `[0 0; 0 0]`

`SubarraySteering`

Subarray steering method

Specify the method of subarray steering as either `'None'` | `'Phase'` | `'Time'` | `'Custom'`.

- When you set this property to `'Phase'`, a phase shifter is used to steer the subarray. Use the `STEERANG` argument of the `step` method to define the steering direction.
- When you set this property to `'Time'`, subarrays are steered using time delays. Use the `STEERANG` argument of the `step` method to define the steering direction.
- When you set this property to `'Custom'`, subarrays are steered by setting independent weights for all elements in each subarray. Use the `WS` argument of the `step` method to define the weights for all subarrays.

Default: `'None'`

`PhaseShifterFrequency`

Subarray phase shifter frequency

Specify the operating frequency of phase shifters that perform subarray steering. The property value is a positive scalar in hertz. This property applies when you set the `SubarraySteering` property to `'Phase'`.

Default: `3e8`

`NumPhaseShifterBits`

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Default: `0`

Methods

Specific to `phased.ReplicatedSubarray` Object	
`beamwidth`	Compute and display beamwidth for a subarray
`collectPlaneWave`	Simulate received plane waves
`directivity`	Directivity of replicated subarray
`getElementPosition`	Positions of array elements
`getNumElements`	Number of elements in array
`getNumSubarrays`	Number of subarrays in array
`getSubarrayPosition`	Positions of subarrays in array
`isPolarizationCapable`	Polarization capability
`pattern`	Plot replicated subarray directivity and patterns
`patternAzimuth`	Plot replicated subarray directivity or pattern versus azimuth
`patternElevation`	Plot replicated subarray directivity or pattern versus elevation
`plotResponse`	Plot response pattern of array
`step`	Output responses of subarrays
`viewArray`	View array geometry

Common to All System Objects	
`release`	Allow System object property value changes

Examples

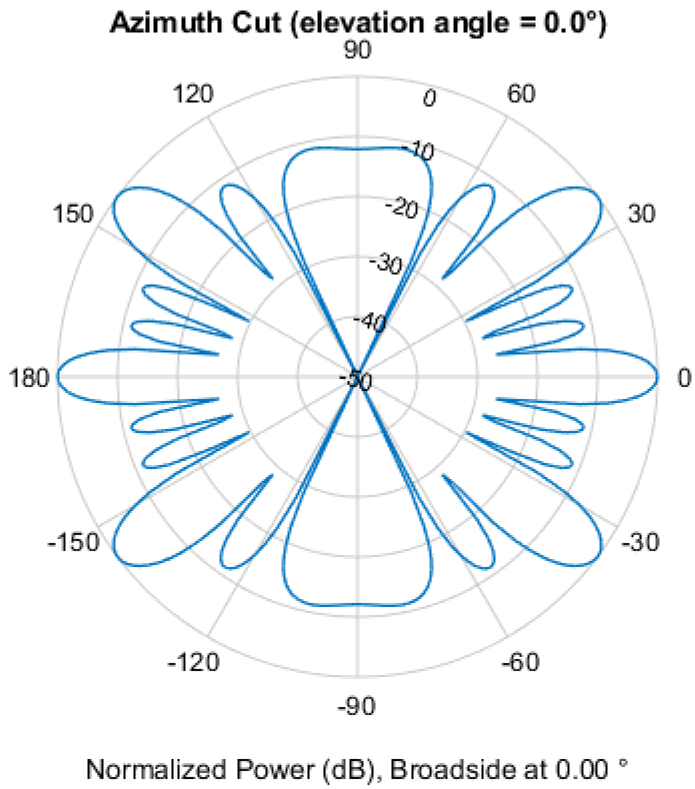
Azimuth Response of Array with Subarrays

Plot the azimuth response of a 4-element ULA composed of two 2-element ULAs. By default, the antenna elements are isotropic.

```
sArray = phased.ULA('NumElements',2,'ElementSpacing',0.5);
sRSA = phased.ReplicatedSubarray('Subarray',sArray,...
   'Layout','Rectangular','GridSize',[1 2],...
   'GridSpacing','Auto');
```

Plot the azimuth response of the array. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
fc = 1.0e9;
pattern(sRSA,fc,[-180:180],0,...
    'PropagationSpeed',physconst('LightSpeed'),...
    'Type','powerdb',...
    'Normalize',true,...
    'CoordinateSystem','polar')
```



Response of Subarrays with Polarized Antenna Elements

Create a 4-element ULA from two 2-element ULA subarrays consisting of short-dipole antenna elements. Then, calculate the response at boresight. Because the array elements support polarization, the response consists of horizontal and vertical components.

Create the arrays from subarrays.

```
sSD = phased.ShortDipoleAntennaElement;
sULA = phased.ULA('Element',sSD,...
```

```
    'NumElements',2,...
    'ElementSpacing',0.5);
sRSA = phased.ReplicatedSubarray('Subarray',sULA,...
    'Layout','Rectangular',...
    'GridSize',[1 2],...
    'GridSpacing','Auto');
```

Show the vertical polarization response for the subarrays.

```
fc = 1.0e9;
ang = [0;0];
resp = step(sRSA,fc,ang,physconst('LightSpeed'));
disp(resp.V)
```

```
   -2.4495
   -2.4495
```

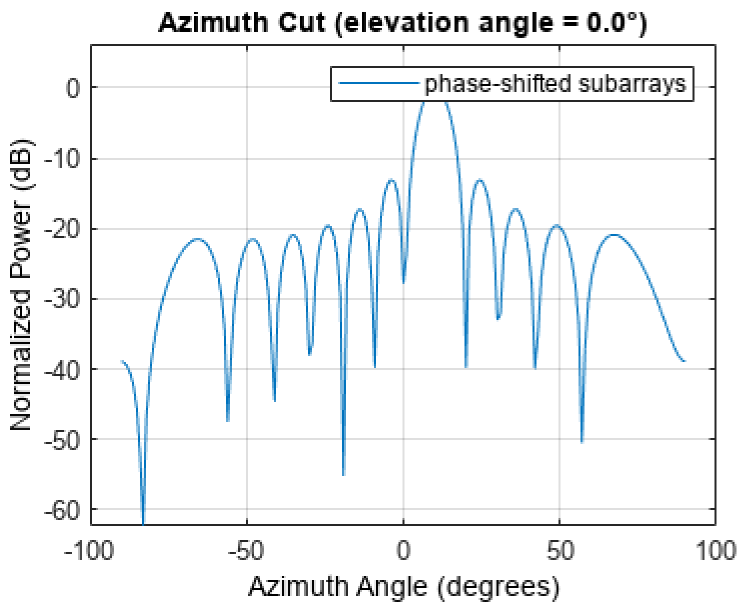
Independently Steered Replicated Subarrays

Create an array consisting of three copies of a 4-element ULA having elements spaced 1/2 wavelength apart. The array operates at 300 MHz.

```
c = physconst('LightSpeed');
fc = 300e6;
lambda = c/fc;
subarray = phased.ULA(4,0.5*lambda);
```

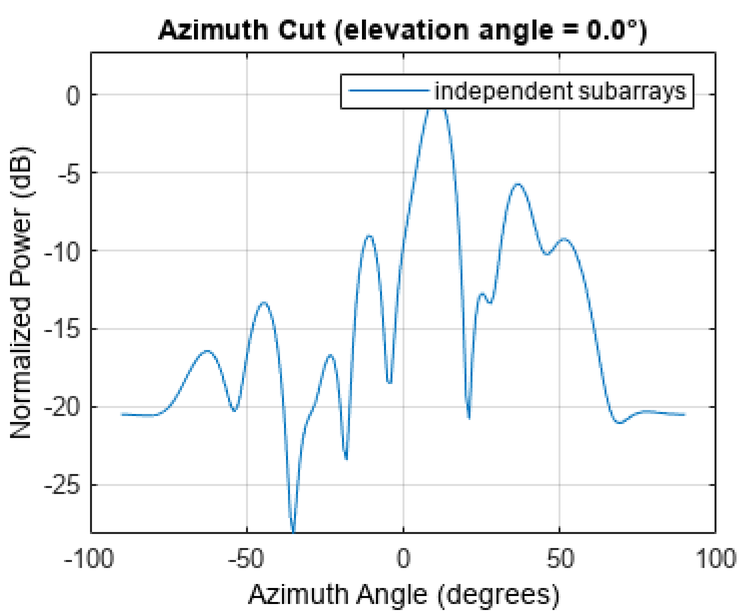
Steer all subarrays by a common phase shift to 10 degrees azimuth.

```
array = phased.ReplicatedSubarray('Subarray',subarray,'GridSize',[1 3], ...
    'SubarraySteering','Phase','PhaseShifterFrequency',fc);
steer_ang = [10;0];
sv_array = phased.SteeringVector('SensorArray',array,...
    'PropagationSpeed',c);
wts_array = sv_array(fc,steer_ang);
pattern(array,fc,-90:90,0,'CoordinateSystem','Rectangular',...
    'Type','powerdb','PropagationSpeed',c,'Weights',wts_array,...
    'SteerAngle',steer_ang);
legend('phase-shifted subarrays')
```



Compute independent subarray weights from subarray steering vectors. The weights point to 5, 15, and 30 degrees azimuth. Set the `SubarraySteering` property to `'Custom'` .

```
steer_ang_subarrays = [5 15 30;0 0 0];
sv_subarray = phased.SteeringVector('SensorArray',subarray,...
    'PropagationSpeed',c);
wc = sv_subarray(fc,steer_ang_subarrays);
array.SubarraySteering = 'Custom';
pattern(array,fc,-90:90,0,'CoordinateSystem','Rectangular',...
    'Type','powerdb','PropagationSpeed',c,'Weights',wts_array,...
    'ElementWeight',conj(wc));
legend('independent subarrays')
hold off
```



Version History

Introduced in R2012a

References

[1] Mailloux, Robert J. *Electronically Scanned Arrays*. San Rafael, CA: Morgan & Claypool Publishers, 2007.

[2] Mailloux, Robert J. *Phased Array Antenna Handbook*, 2nd Ed. Norwood, MA: Artech House, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, `plotResponse`, and `viewArray` methods are not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`phased.ULA` | `phased.URA` | `phased.UCA` | `phased.ConformalArray` | `phased.PartitionedArray`

Apps

Sensor Array Analyzer

Topics

Subarrays in Phased Array Antennas
Phased Array Gallery
“Subarrays Within Arrays”

directivity

System object: phased.ReplicatedSubarray
Package: phased

Directivity of replicated subarray

Syntax

```
D = directivity(H,FREQ,ANGLE)
D = directivity(H,FREQ,ANGLE,Name,Value)
```

Description

`D = directivity(H,FREQ,ANGLE)` returns the “Directivity (dBi)” on page 1-1437 of a replicated array of antenna or microphone element, `H`, at frequencies specified by `FREQ` and in angles of direction specified by `ANGLE`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`D = directivity(H,FREQ,ANGLE,Name,Value)` returns the directivity with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

H — Replicated subarray
System object

Replicated subarray, specified as a `phased.ReplicatedSubarray` System object.

Example: `H = phased.ReplicatedSubarray;`

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

ANGLE — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

PropagationSpeed — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Subarray weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Subarray weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*M* complex-valued matrix. The dimension *N* is the number of subarrays in the array. The dimension *L* is the number of frequencies specified by the `FREQ` argument.

`Weights` dimension	`FREQ` dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in the `FREQ` argument.

Example: `'Weights',ones(N,M)`

Data Types: `double`

`SteerAngle` — Subarray steering angle
`[0;0]` (default) | scalar | 2-element column vector

Subarray steering angle, specified as the comma-separated pair consisting of `'SteerAngle'` and a scalar or a 2-by-1 column vector.

If `'SteerAngle'` is a 2-by-1 column vector, it has the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `'SteerAngle'` is a scalar, it specifies the azimuth angle only. In this case, the elevation angle is assumed to be 0.

This option applies only when the `'SubarraySteering'` property of the System object is set to `'Phase'` or `'Time'`.

Example: `'SteerAngle',[20;30]`

Data Types: `double`

`ElementWeights` — Weights applied to elements within subarray
`1` (default) | complex-valued N_{SE}-by-N matrix

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix. Weights are applied to the individual elements within a subarray. All subarrays have the same dimensions and sizes. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray.

Dependencies

To enable this name-value pair, set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`D` — Directivity
M-by-L matrix

Directivity, returned as an M-by-L matrix. Each row corresponds to one of the M angles specified by `ANGLE`. Each column corresponds to one of the L frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Replicated Subarray

Compute the directivity of an array built up from ULA subarrays. Determine the directivity of the replicated subarray when the array is steered to towards 30 degrees azimuth.

Set the signal propagation speed to the speed of light. Set the signal frequency to 300 MHz.

```
c = physconst('LightSpeed');
fc = 3e8;
lambda = c/fc;
```

Create a 4-element ULA of isotropic antenna elements spaced 0.4-wavelength apart.

```
myArray = phased.ULA;
myArray.NumElements = 4;
myArray.ElementSpacing = 0.4*lambda;
```

Construct a 2-by-1 replicated subarray.

```
myRepArray = phased.ReplicatedSubarray;
myRepArray.Subarray = myArray;
myRepArray.Layout = 'Rectangular';
myRepArray.GridSize = [2 1];
myRepArray.GridSpacing = 'Auto';
myRepArray.SubarraySteering = 'Time';
```

Steer the array to 30 degrees azimuth and zero degrees elevation.

```
ang = [30;0];
mySV = phased.SteeringVector;
mySV.SensorArray = myRepArray;
mySV.PropagationSpeed = c;
```

Find the directivity at 30 degrees azimuth.

```
d = directivity(myRepArray,fc,ang,...
    'PropagationSpeed',c,...
    'Weights',step(mySV,fc,ang),...
    'SteerAngle',ang)
```

```
d = 7.4776
```

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`patternElevation` | `patternAzimuth` | `pattern`

collectPlaneWave

System object: `phased.ReplicatedSubarray`
Package: `phased`

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

`Y = collectPlaneWave(H,X,ANG)` returns the received signals at the sensor array, `H`, when the input signals indicated by `X` arrive at the array from the directions specified in `ANG`.

`Y = collectPlaneWave(H,X,ANG,FREQ)`, in addition, specifies the incoming signal carrier frequency in `FREQ`.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)`, in addition, specifies the signal propagation speed in `C`.

Input Arguments

H

Array object.

X

Incoming signals, specified as an M-column matrix. Each column of `X` represents an individual incoming signal.

ANG

Directions from which incoming signals arrive, in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column specifies the direction of arrival of the corresponding signal in `X`. Each column of `ANG` is in the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `ANG` is a row vector of length M, each entry in `ANG` specifies the azimuth angle. In this case, the corresponding elevation angle is assumed to be 0°.

FREQ

Carrier frequency of signal in hertz. `FREQ` must be a scalar.

Default: `3e8`

c

Propagation speed of signal in meters per second.

Default: Speed of light

Output Arguments

Y

Received signals. Y is an N-column matrix, where N is the number of subarrays in the array H. Each column of Y is the received signal at the corresponding subarray, with all incoming signals combined.

Examples

Plane Waves Received at Array of Subarrays

Simulate the received signal at a 16-element ULA composed of four 4-element ULAs.

```
array = phased.ULA('NumElements',4);
subarrays = phased.ReplicatedSubarray('Subarray',array,'GridSize',[4 1]);
```

Simulate two signals received from 10° azimuth and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
y = collectPlaneWave(subarrays,randn(4,2),[10 30],100.0e6,...
    physconst('LightSpeed'));
```

Algorithms

`collectPlaneWave` modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. This method does not account for the response of individual elements in the array and only models the array factor among subarrays. Therefore, the result does not depend on whether the subarray is steered.

See Also

`uv2azel` | `phitheta2azel`

getElementPosition

System object: phased.ReplicatedSubarray
Package: phased

Positions of array elements

Syntax

```
POS = getElementPosition(H)
```

Description

`POS = getElementPosition(H)` returns the element positions in the array `H`.

Input Arguments

H

Array object consisting of replicated subarrays.

Output Arguments

POS

Element positions in array. `POS` is a 3-by-N matrix, where N is the number of elements in `H`. Each column of `POS` defines the position of an element in the local coordinate system, in meters, using the form [x; y; z].

Examples

Positions of Elements in Array with Replicated Subarrays

Create an array with two copies of a 3-element ULA, and obtain the positions of the elements.

```
subarrays = phased.ReplicatedSubarray('Subarray',...
    phased.ULA('NumElements',3),'GridSize',[1 2]);
pos = getElementPosition(subarrays)
```

```
pos = 3×6

         0         0         0         0         0         0
   -1.2500   -0.7500   -0.2500    0.2500    0.7500    1.2500
         0         0         0         0         0         0
```

See Also

getSubarrayPosition

getNumElements

System object: phased.ReplicatedSubarray
Package: phased

Number of elements in array

Syntax

N = getNumElements(H)

Description

N = getNumElements(H) returns the number of elements in the array object H. This number includes the elements in all subarrays of the array.

Input Arguments

H

Array object consisting of replicated subarrays.

Examples

Number of Elements in Array with Replicated Subarrays

Create an array with two copies of a 3-element ULA, and obtain the total number of elements.

```
subarray = phased.ReplicatedSubarray('Subarray',...
    phased.ULA('NumElements',3),'GridSize',[1 2]);
N = getNumElements(subarray)
```

```
N = 6
```

See Also

getNumSubarrays

getNumSubarrays

System object: `phased.ReplicatedSubarray`
Package: `phased`

Number of subarrays in array

Syntax

`N = getNumSubarrays(H)`

Description

`N = getNumSubarrays(H)` returns the number of subarrays in the array object `H`.

Input Arguments

`H`

Array object consisting of replicated subarrays.

Examples

Number of Subarrays in Array

Create an array by tiling copies of a ULA in a 2-by-5 grid. Then, obtain the number of subarrays.

```
subarrays = phased.ReplicatedSubarray('Subarray',...
    phased.ULA('NumElements',3),'GridSize',[2 5]);
N = getNumSubarrays(subarrays)
```

```
N = 10
```

See Also

`getNumElements`

getSubarrayPosition

System object: `phased.ReplicatedSubarray`
Package: `phased`

Positions of subarrays in array

Syntax

`POS = getSubarrayPosition(H)`

Description

`POS = getSubarrayPosition(H)` returns the subarray positions in the array `H`.

Input Arguments

H

Partitioned array object.

Output Arguments

POS

Subarrays positions in array. `POS` is a 3-by-N matrix, where N is the number of subarrays in `H`. Each column of `POS` defines the position of a subarray in the local coordinate system, in meters, using the form [x; y; z].

Examples

Replicated Subarray Positions

Create an array with two copies of a 3-element ULA, and obtain the positions of the subarrays.

```
subarray = phased.ReplicatedSubarray('Subarray',...
    phased.ULA('NumElements',3),'GridSize',[1 2]);
pos = getSubarrayPosition(subarray)
```

```
pos = 3×2

         0         0
   -0.7500    0.7500
         0         0
```

See Also
`getElementPosition`

isPolarizationCapable

System object: phased.ReplicatedSubarray
Package: phased

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h — Replicated subarray

Replicated subarray specified as a `phased.ReplicatedSubarray` System object.

Output Arguments

flag — Polarization-capability flag

Polarization-capability flag returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

Replicated Array of Short Dipoles Supports Polarization

Verify that a replicated subarray of short-dipole antenna elements supports polarization.

```
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[1e9 10e9]);
array = phased.URA([3,2],'Element',antenna);
reparray = phased.ReplicatedSubarray('Subarray',array, ...
    'Layout','Rectangular','GridSize',[1,2],'GridSpacing','Auto');
isPolarizationCapable(reparray)
```

```
ans = logical
   1
```

pattern

System object: `phased.ReplicatedSubarray`
Package: `phased`

Plot replicated subarray directivity and patterns

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sArray,FREQ)` plots the 3-D array directivity pattern (in dBi) for the array specified in `sArray`. The operating frequency is specified in `FREQ`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`pattern(sArray,FREQ,AZ)` plots the array directivity pattern at the specified azimuth angle.

`pattern(sArray,FREQ,AZ,EL)` plots the array directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the array pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See "Convert plotResponse to pattern" on page 1-1454 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sArray` — Replicated subarray
System object

Replicated subarray, specified as a `phased.ReplicatedSubarray` System object.

Example: `sArray= phased.ReplicatedSubarray;`

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of 'CoordinateSystem' and one of 'polar', 'rectangular', or 'uv'. When 'CoordinateSystem' is set to 'polar' or 'rectangular', the AZ and EL arguments specify the pattern azimuth and elevation, respectively. AZ values must lie between –180° and 180°. EL values must lie between –90° and 90°. If 'CoordinateSystem' is set to 'uv', AZ and EL then specify *U* and *V* coordinates, respectively. AZ and EL must lie between -1 and 1.

Example: 'uv'

Data Types: char

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Orientation — Array orientation
[0;0;0]. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

ShowArray — View array geometry
false (default) | true

View the array geometry along with the 3D radiation pattern, specified as false or true.

Data Types: logical

ShowLocalCoordinates — Show local coordinate axes
true (default) | false

Show the local coordinate axes, specified as true or false.

Data Types: logical

ShowColorbar — Show colorbar
true (default) | false

Show the colorbar, specified as `true` or `false`.

Data Types: `logical`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Subarray weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Subarray weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*M* complex-valued matrix. The dimension *N* is the number of subarrays in the array. The dimension *L* is the number of frequencies specified by the `FREQ` argument.

`Weights` dimension	`FREQ` dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `‘Weights’` for the corresponding frequency in the `FREQ` argument.

Example: `'Weights',ones(N,M)`

Data Types: `double`

`SteerAngle` — Subarray steering angle
`[0;0]` (default) | scalar | 2-element column vector

Subarray steering angle, specified as the comma-separated pair consisting of `'SteerAngle'` and a scalar or a 2-by-1 column vector.

If `'SteerAngle'` is a 2-by-1 column vector, it has the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `'SteerAngle'` is a scalar, it specifies the azimuth angle only. In this case, the elevation angle is assumed to be 0.

This option applies only when the `'SubarraySteering'` property of the System object is set to `'Phase'` or `'Time'`.

Example: `'SteerAngle',[20;30]`

Data Types: `double`

`ElementWeights` — Weights applied to elements within subarray
`1` (default) | complex-valued N_{SE}-by-N matrix

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix. Weights are applied to the individual elements within a subarray. All subarrays have the same dimensions and sizes. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray.

Dependencies

To enable this name-value pair, set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`PAT` — Array pattern
M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of `PAT` correspond to the dimensions of the output arguments `AZ_ANG` and `EL_ANG`.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

Examples

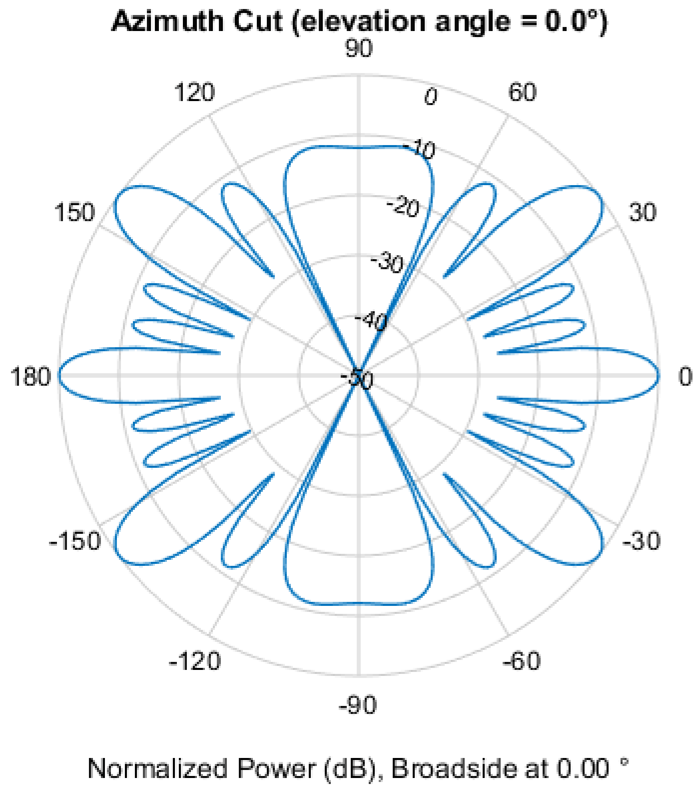
Azimuth Response of Array with Subarrays

Plot the azimuth response of a 4-element ULA composed of two 2-element ULAs. By default, the antenna elements are isotropic.

```
sArray = phased.ULA('NumElements',2,'ElementSpacing',0.5);
sRSA = phased.ReplicatedSubarray('Subarray',sArray,...
   'Layout','Rectangular','GridSize',[1 2],...
   'GridSpacing','Auto');
```

Plot the azimuth response of the array. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
fc = 1.0e9;
pattern(sRSA,fc,[-180:180],0,...
    'PropagationSpeed',physconst('LightSpeed'),...
    'Type','powerdb',...
    'Normalize',true,...
    'CoordinateSystem','polar')
```



Directivity of Array with Subarrays

Create a 2-by-2-element URA of isotropic antenna elements, and arrange four copies to form a 16-element URA. Plot the 3-D directivity pattern.

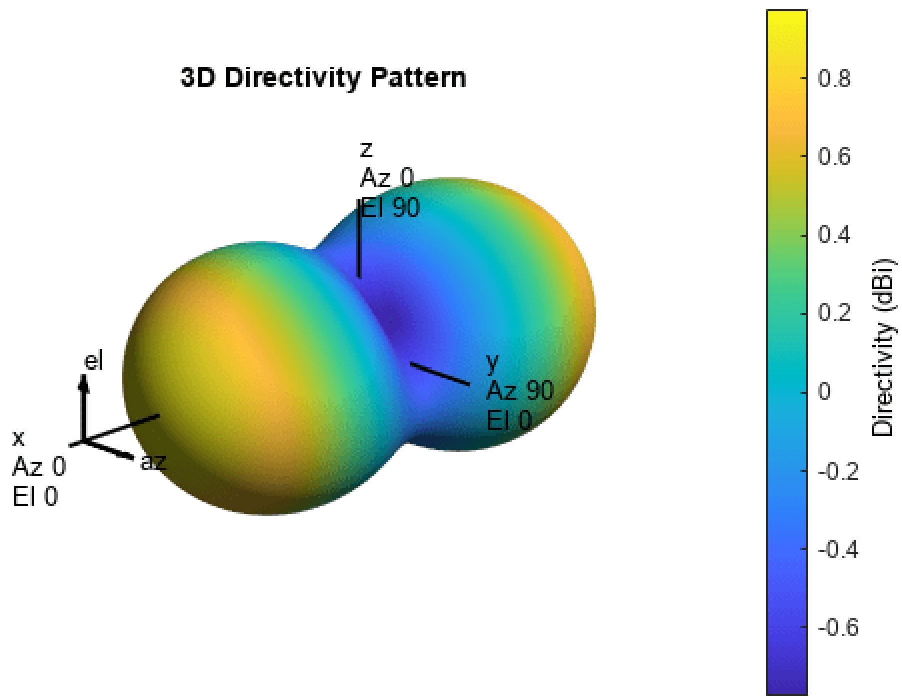
Create the array

```
fmin = 1e9;
fmax = 6e9;
c = physconst('LightSpeed');
lam = c/fmax;
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[fmin,fmax],...
    'BackBaffled',false);
sURA = phased.URA('Element',sIso,...
    'Size',[2 2],...
    'ElementSpacing',lam/2);
sRS = phased.ReplicatedSubarray('Subarray',sURA,...
    'Layout','Rectangular','GridSize',[2 2],...
    'GridSpacing','Auto');
```

Plot 3-D directivity pattern

```
fc = 1e9;
wts = [0.862,1.23,1.23,0.862]';
```

```
pattern(sRS,fc,[-180:180],[-90:90],...
    'PropagationSpeed',physconst('LightSpeed'),....
    'Type','directivity',...
    'Weights',wts);
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.

Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.

Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.

'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the UV grid values using AZ and EL.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
'ElevationAngles' name-value pair	Elevation angles used to display the antenna or array response.	EL argument
'UGrid' name-value pair	Contains *U* coordinates in *UV*-space.	AZ argument when 'CoordinateSystem' name-value pair is set to 'uv'
'VGrid' name-value pair	Contains *V*-coordinates in *UV*-space.	EL argument when 'CoordinateSystem' name-value pair is set to 'uv'

Version History

Introduced in R2015a

See Also

patternElevation | patternAzimuth

patternAzimuth

System object: `phased.ReplicatedSubarray`
Package: `phased`

Plot replicated subarray directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sArray,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(sArray,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sArray,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sArray` — Replicated subarray
System object

Replicated subarray, specified as a `phased.ReplicatedSubarray` System object.

Example: `sArray= phased.ReplicatedSubarray;`

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Subarray weights
M-by-1 complex-valued column vector

Subarray weights, specified as the comma-separated pair consisting of 'Weights' and an M-by-1 complex-valued column vector. Subarray weights are applied to the subarrays of the array to produce array steering, tapering, or both. The dimension M is the number of subarrays in the array.

Example: 'Weights',ones(10,1)

Data Types: double
Complex Number Support: Yes

SteerAngle — Subarray steering angle
[0;0] (default) | scalar | 2-element column vector

Subarray steering angle, specified as the comma-separated pair consisting of 'SteerAngle' and a scalar or a 2-by-1 column vector.

If 'SteerAngle' is a 2-by-1 column vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If 'SteerAngle' is a scalar, it specifies the azimuth angle only. In this case, the elevation angle is assumed to be 0.

This option applies only when the 'SubarraySteering' property of the System object is set to 'Phase' or 'Time'.

Example: 'SteerAngle',[20;30]

Data Types: double

ElementWeights — Weights applied to elements within subarray
1 (default) | complex-valued N_{SE}-by-N matrix

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix. Weights are applied to the individual elements within a subarray. All subarrays have the same dimensions and sizes. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray.

Dependencies

To enable this name-value pair, set the SubarraySteering property of the array to 'Custom'.

Data Types: double
Complex Number Support: Yes

Azimuth — Azimuth angles
[-180:180] (default) | 1-by-P real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of 'Azimuth' and a 1-by-P real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: 'Azimuth',[-90:2:90]

Data Types: double

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an L-by-N real-valued matrix. The dimension L is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension N is the number of elevation angles, as determined by the `EL` input argument.

Examples

Azimuth Pattern of Array with Subarrays

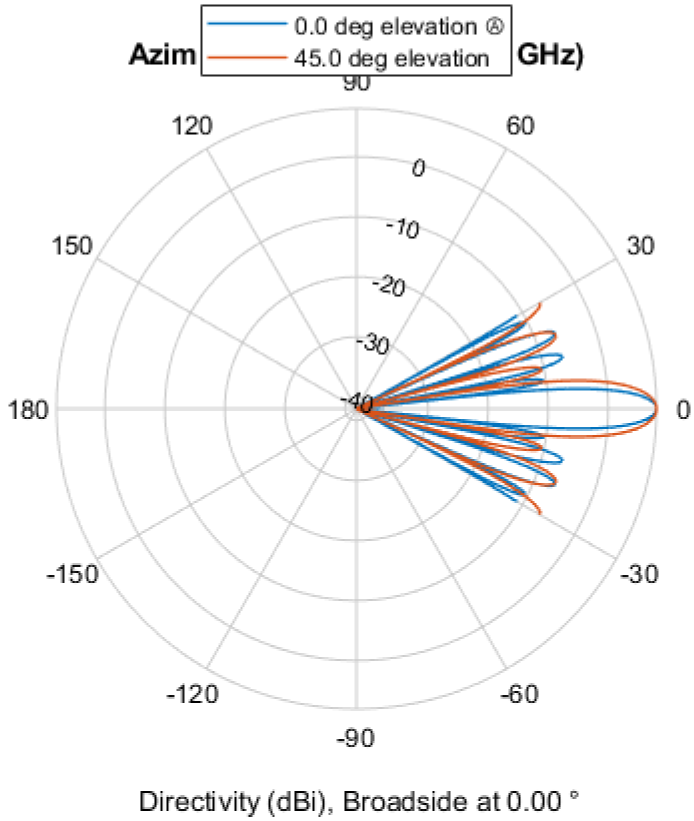
Create a 2-element ULA of isotropic antenna elements, and arrange three copies to form a 6-element ULA. Plot the directivity azimuth pattern within a restricted range of azimuth angles from -30 to 30 degrees in 0.1 degree increments. Plot directivity for 0 degrees and 45 degrees elevation.

Create the array

```
fmin = 1e9;
fmax = 6e9;
c = physconst('LightSpeed');
lam = c/fmax;
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[fmin,fmax],...
    'BackBaffled',false);
sULA = phased.ULA('Element',sIso,...
    'NumElements',2,'ElementSpacing',0.5);
sRS = phased.ReplicatedSubarray('Subarray',sULA,...
    'Layout','Rectangular','GridSize',[1 3],...
    'GridSpacing','Auto');
```

Plot azimuth directivity pattern

```
fc = 1e9;
wts = [0.862,1.23,0.862]';
patternAzimuth(sRS,fc,[0,45],'PropagationSpeed',physconst('LightSpeed'),...
    'Azimuth',[-30:0.1:30],...
    'Type','directivity',...
    'Weights',wts);
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also
patternElevation | pattern

patternElevation

System object: `phased.ReplicatedSubarray`
Package: `phased`

Plot replicated subarray directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sArray,FREQ)` plots the 2-D array directivity pattern versus elevation (in dBi) for the array `sArray` at zero degrees azimuth angle. When `AZ` is a vector, multiple overlaid plots are created. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternElevation(sArray,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sArray,FREQ,AZ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sArray` — Replicated subarray
System object

Replicated subarray, specified as a `phased.ReplicatedSubarray` System object.

Example: `sArray= phased.ReplicatedSubarray;`

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Subarray weights
M-by-1 complex-valued column vector

Subarray weights, specified as the comma-separated pair consisting of 'Weights' and an M-by-1 complex-valued column vector. Subarray weights are applied to the subarrays of the array to produce array steering, tapering, or both. The dimension M is the number of subarrays in the array.

Example: 'Weights',ones(10,1)

Data Types: double
Complex Number Support: Yes

SteerAngle — Subarray steering angle
[0;0] (default) | scalar | 2-element column vector

Subarray steering angle, specified as the comma-separated pair consisting of 'SteerAngle' and a scalar or a 2-by-1 column vector.

If 'SteerAngle' is a 2-by-1 column vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If 'SteerAngle' is a scalar, it specifies the azimuth angle only. In this case, the elevation angle is assumed to be 0.

This option applies only when the 'SubarraySteering' property of the System object is set to 'Phase' or 'Time'.

Example: 'SteerAngle',[20;30]

Data Types: double

ElementWeights — Weights applied to elements within subarray
1 (default) | complex-valued N_{SE}-by-N matrix

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix. Weights are applied to the individual elements within a subarray. All subarrays have the same dimensions and sizes. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray.

Dependencies

To enable this name-value pair, set the SubarraySteering property of the array to 'Custom'.

Data Types: double
Complex Number Support: Yes

Elevation — Elevation angles
[-90:90] (default) | 1-by-P real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of 'Elevation' and a 1-by-P real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: 'Elevation',[-90:2:90]

Data Types: double

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Elevation Pattern of Array with Subarrays

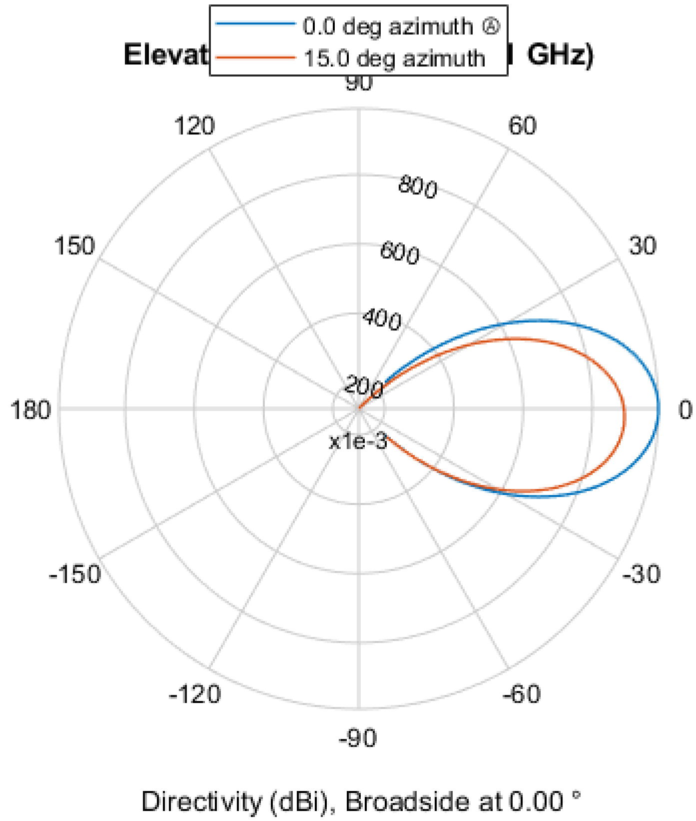
Create a 2-by-2-element URA of isotropic antenna elements, and arrange four copies to form a 16-element URA. Plot the elevation directivity pattern within a restricted range of elevation angles from -45 to 45 degrees in 0.1 degree increments. Plot directivity for 0 degrees and 15 degrees azimuth.

Create the array

```
fmin = 1e9;
fmax = 6e9;
c = physconst('LightSpeed');
lam = c/fmax;
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[fmin,fmax],...
    'BackBaffled',false);
sURA = phased.URA('Element',sIso,...
    'Size',[2 2],...
    'ElementSpacing',lam/2);
sRS = phased.ReplicatedSubarray('Subarray',sURA,...
    'Layout','Rectangular','GridSize',[2 2],...
    'GridSpacing','Auto');
```

Plot elevation directivity pattern

```
fc = 1e9;
wts = [0.862,1.23,1.23,0.862]';
patternElevation(sRS,fc,[0,15],...
    'PropagationSpeed',physconst('LightSpeed'),...
    'Elevation',[-45:0.1:45],...
    'Type','directivity',...
    'Weights',wts);
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{rad}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `pattern`

plotResponse

System object: `phased.ReplicatedSubarray`
Package: `phased`

Plot response pattern of array

Syntax

```
plotResponse(H,FREQ,V)
plotResponse(H,FREQ,V,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ,V)` plots the array response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`. The propagation speed is specified in `V`.

`plotResponse(H,FREQ,V,Name,Value)` plots the array response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

`H`

Array object.

`FREQ`

Operating frequency, in hertz. Typical values are within the range specified by a property of `H.Subarray.Element`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has zero response at frequencies outside that range. If `FREQ` is a nonscalar row vector, the plot shows multiple frequency responses on the same axes.

`V`

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

CutAngle

Cut angle specified as a scalar. This argument is applicable only when RespCut is 'Az' or 'El'. If RespCut is 'Az', CutAngle must be between –90 and 90. If RespCut is 'El', CutAngle must be between –180 and 180.

Default: 0

Format

Format of the plot, using one of 'Line', 'Polar', or 'UV'. If you set Format to 'UV', FREQ must be a scalar.

Default: 'Line'

NormalizeResponse

Set this value to true to normalize the response pattern. Set this value to false to plot the response pattern without normalizing it. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: true

OverlayFreq

Set this value to true to overlay pattern cuts in a 2-D line plot. Set this value to false to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is false, then FREQ must be a vector with at least two entries.

This parameter applies only when Format is not 'Polar' and RespCut is not '3D'.

Default: true

Polarization

Specify the polarization options for plotting the array response pattern. The allowable values are |'None' | 'Combined' | 'H' | 'V' | where:

- 'None' specifies plotting a nonpolarized response pattern
- 'Combined' specifies plotting a combined polarization response pattern
- 'H' specifies plotting the horizontal polarization response pattern
- 'V' specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is 'None'. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: 'None'

RespCut

Cut of the response. Valid values depend on Format, as follows:

- If Format is 'Line' or 'Polar', the valid values of RespCut are 'Az', 'El', and '3D'. The default is 'Az'.

- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

`SteerAng`

Subarray steering angle. `SteerAng` can be either a 2-element column vector or a scalar.

If `SteerAng` is a 2-element column vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If `SteerAng` is a scalar, it specifies the azimuth angle. In this case, the elevation angle is assumed to be 0.

This option is applicable only if the `SubarraySteering` property of `H` is `'Phase'` or `'Time'`.

Default: `[0;0]`

`Unit`

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

`Weights`

Weight values applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. The dimension *N* is the number of subarrays in the array. The interpretation of *M* depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

AzimuthAngles

Azimuth angles for plotting subarray response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

ElevationAngles

Elevation angles for plotting subarray response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

UGrid

U coordinate values for plotting subarray response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

VGrid

V coordinate values for plotting subarray response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Azimuth Response and Directivity of ULA with Subarrays

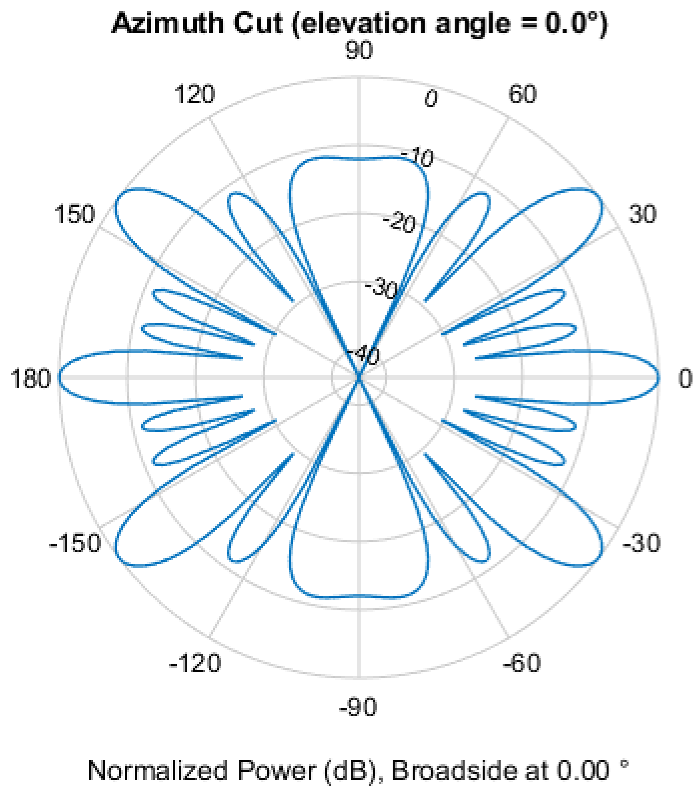
Plot the azimuth response of a 4-element ULA composed of two 2-element ULAs.

Create a 2-element ULA, and arrange two copies to form a 4-element ULA.

```
h = phased.ULA('NumElements',2,'ElementSpacing',0.5);
ha = phased.ReplicatedSubarray('Subarray',h,...
    'Layout','Rectangular','GridSize',[1 2],...
    'GridSpacing','Auto');
```

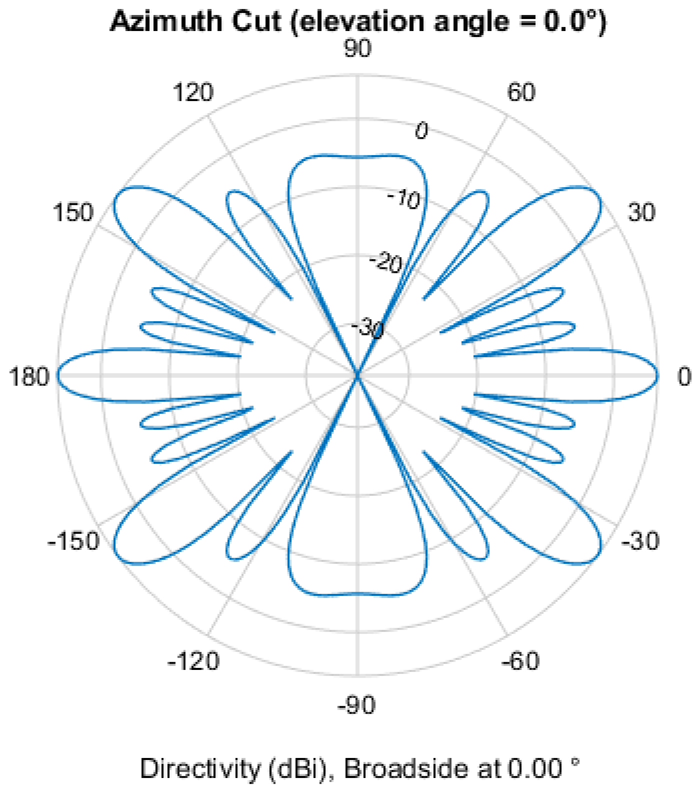
Plot the azimuth response of the array. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

```
plotResponse(ha,1e9,3e8,'RespCut','Az','Format','Polar');
```



Plot the azimuth directivity of the array.

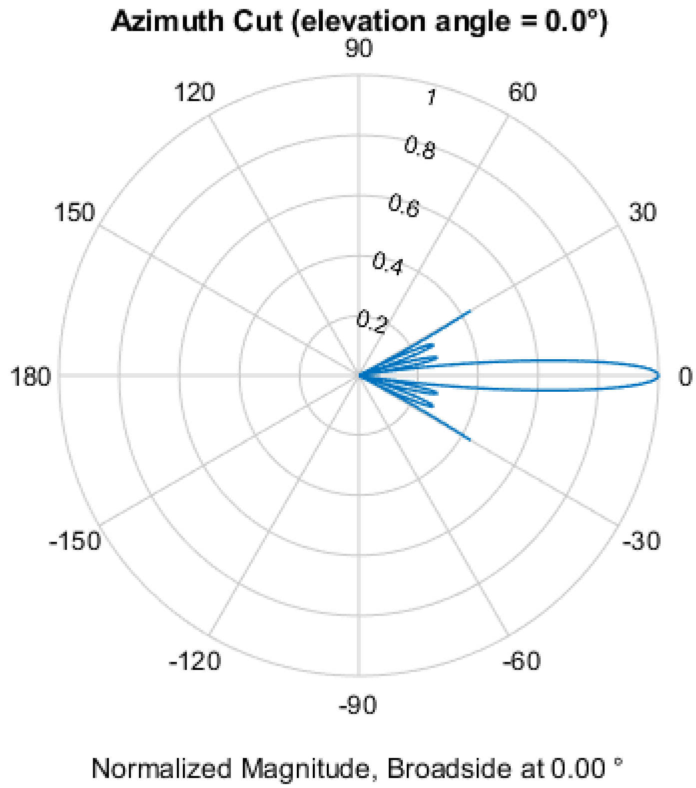
```
plotResponse(ha,1e9,3e8,'RespCut','Az','Format','Polar','Unit','dbi');
```



Display Azimuth Response of Array with Subarrays Between -30 and 30 Degrees

Create a 2-element ULA, and arrange two copies to form a 4-element ULA. Using the `AzimuthAngles` parameter, plot the response within a restricted range of azimuth angles from -30 to 30 degrees in 0.1 degree increments.

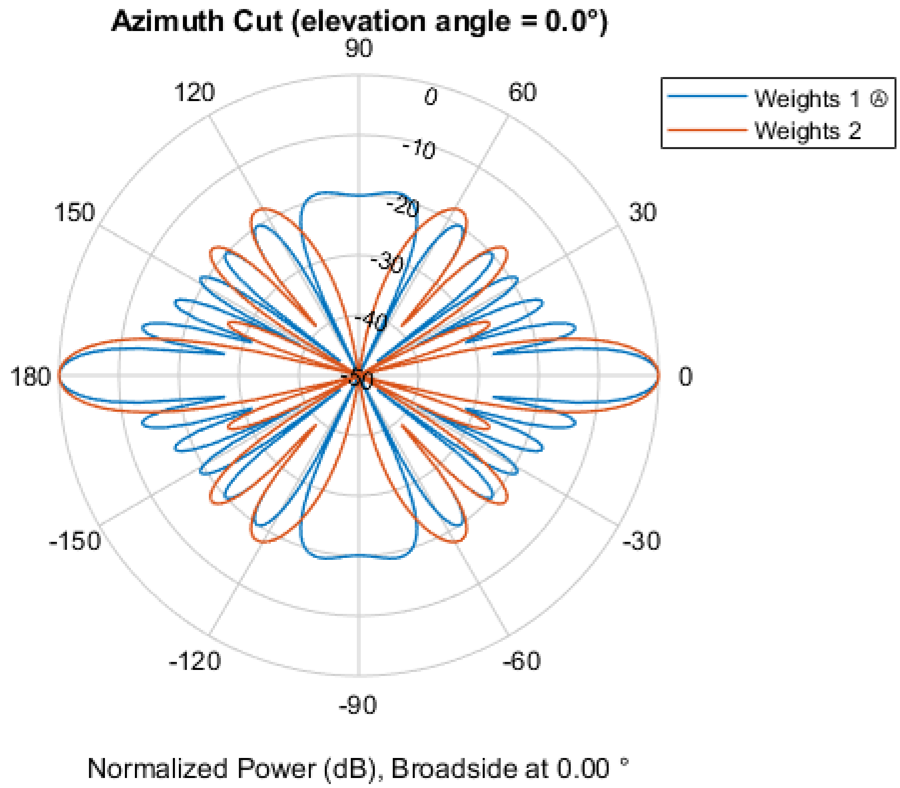
```
h = phased.ULA('NumElements',2,'ElementSpacing',0.5);
ha = phased.ReplicatedSubarray('Subarray',h,...
    'Layout','Rectangular','GridSize',[1 2],...
    'GridSpacing','Auto');
plotResponse(ha,1e9,3e8,'RespCut','Az','Format','Polar',...
    'AzimuthAngles',[-30:0.1:30],'Unit','mag');
```



Apply Two Sets of Weights at a Single Frequency

Construct an array of replicated subarrays. Start with a 2-element uniform line array (ULA), and duplicate it 5 times to create a 10-element ULA. Apply both uniform weights and tapered weights. Then, use `plotResponse` to show that the tapered set of weights reduces the adjacent sidelobes while broadening the main lobe.

```
h = phased.ULA('NumElements',2,'ElementSpacing',0.2);
ha = phased.ReplicatedSubarray('Subarray',h,...
    'Layout','Rectangular','GridSize',[1 5],...
    'GridSpacing',0.4);
c = physconst('LightSpeed');
fc = 1e9;
wts1 = [0.2,0.2,0.2,0.2,0.2]';
wts2 = [0.1,0.23333,.33333,0.23333,0.1]';
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar',...
    'Weights',[wts1,wts2]);
```



See Also

`uv2azel` | `azel2uv`

step

System object: `phased.ReplicatedSubarray`
Package: `phased`

Output responses of subarrays

Syntax

```
RESP = step(H,FREQ,ANG,V)
RESP = step(H,FREQ,ANG,V,STEERANGLE)
RESP = step(H,FREQ,ANG,V,WS)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG,V)` returns the responses, `RESP`, of the subarrays in the array, at operating frequencies specified in `FREQ` and directions specified in `ANG`. `V` is the propagation speed. The elements within each subarray are connected to the subarray phase center using an equal-path feed.

`RESP = step(H,FREQ,ANG,V,STEERANGLE)` uses `STEERANGLE` as the steering direction of the subarray. This syntax is available when you set the `SubarraySteering` property to either `'Phase'` or `'Time'`.

`RESP = step(H,FREQ,ANG,V,WS)` uses `WS` as the subarray element weights. This syntax is available when you set the `SubarraySteering` property to `'Custom'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Phased array formed by replicated subarrays.

FREQ

Operating frequencies of array in hertz. `FREQ` is a row vector of length L. Typical values are within the range specified by a property of `H.Subarray.Element`. That property is named

FrequencyRange or FrequencyVector, depending on the type of element in the array. The element has zero response at frequencies outside that range.

ANG

Directions in degrees. ANG can be either a 2-by-M matrix or a row vector of length M.

If ANG is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If ANG is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

V

Propagation speed in meters per second. This value must be a scalar.

STEERANGLE

Subarray steering direction. STEERANGLE can be either a 2-element column vector or a scalar.

If this argument is a 2-element column vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If STEERANGLE is a scalar, it specifies the direction's azimuth angle. In this case, the elevation angle is assumed to be 0°.

Dependencies

To enable this argument, set the SubarraySteering to 'Phase' or 'Time'.

WS

Subarray element weights

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix. Weights are applied to the individual elements within a subarray. All subarrays have the same dimensions and sizes. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of the matrix specifies the weights for the corresponding subarray.

Dependencies

To enable this argument, set the SubarraySteering to 'Custom'.

Output Arguments

RESP

Voltage responses of the subarrays of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, RESP, has the dimensions N-by-M-by-L. The first dimension, N , represents the number of subarrays in the phased array, the second dimension, M, represents the number of angles specified in ANG, while L

represents the number of frequencies specified in FREQ. Each column of RESP contains the responses of the subarrays for the corresponding direction specified in ANG. Each of the L pages of RESP contains the responses of the subarrays for the corresponding frequency specified in FREQ.

- If the array is capable of supporting polarization, the voltage response, RESP, is a MATLAB struct containing two fields, RESP.H and RESP.V, each having dimensions N-by-M-by-L. The field, RESP.H, represents the array's horizontal polarization response, while RESP.V represents the array's vertical polarization response. The first dimension, N , represents the number of subarrays in the phased array, the second dimension, M, represents the number of angles specified in ANG, while L represents the number of frequencies specified in FREQ. Each of the M columns contains the responses of the subarrays for the corresponding direction specified in ANG. Each of the L pages contains the responses of the subarrays for the corresponding frequency specified in FREQ.

Examples

Subarray Response

Calculate the response at boresight for two 2-element ULA arrays that form subarrays of a 4-element ULA array of short-dipole antenna elements.

Create a two-element ULA of short-dipole antenna elements. Then, arrange two copies to form a 4-element ULA.

```
antenna = phased.ShortDipoleAntennaElement;
array = phased.ULA('Element',antenna,'NumElements',2,'ElementSpacing',0.5);
replicatedarray = phased.ReplicatedSubarray('Subarray',array,...
    'Layout','Rectangular','GridSize',[1 2],...
    'GridSpacing','Auto');
```

Find the response of each subarray at boresight. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
c = physconst('LightSpeed');
resp = replicatedarray(1.0e9,[0;0],c)
```

```
resp = struct with fields:
    H: [2x1 double]
    V: [2x1 double]
```

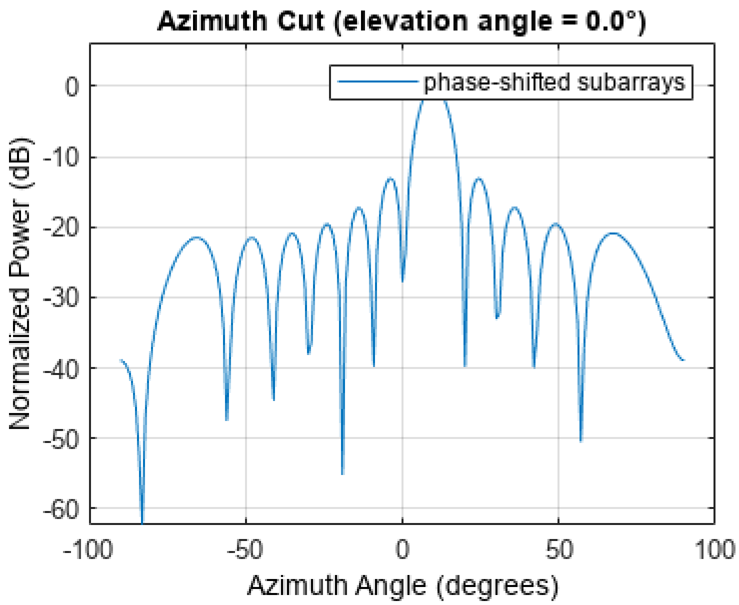
Independently Steered Replicated Subarrays

Create an array consisting of three copies of a 4-element ULA having elements spaced 1/2 wavelength apart. The array operates at 300 MHz.

```
c = physconst('LightSpeed');
fc = 300e6;
lambda = c/fc;
subarray = phased.ULA(4,0.5*lambda);
```

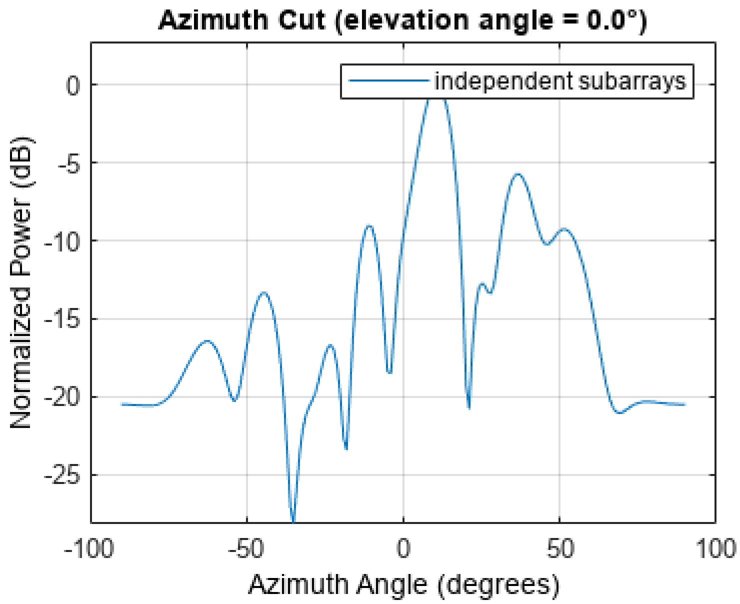
Steer all subarrays by a common phase shift to 10 degrees azimuth.

```
array = phased.ReplicatedSubarray('Subarray',subarray,'GridSize',[1 3], ...
    'SubarraySteering','Phase','PhaseShifterFrequency',fc);
steer_ang = [10;0];
sv_array = phased.SteeringVector('SensorArray',array,...
    'PropagationSpeed',c);
wts_array = sv_array(fc,steer_ang);
pattern(array,fc,-90:90,0,'CoordinateSystem','Rectangular',...
    'Type','powerdb','PropagationSpeed',c,'Weights',wts_array,...
    'SteerAngle',steer_ang);
legend('phase-shifted subarrays')
```



Compute independent subarray weights from subarray steering vectors. The weights point to 5, 15, and 30 degrees azimuth. Set the `SubarraySteering` property to `'Custom'` .

```
steer_ang_subarrays = [5 15 30;0 0 0];
sv_subarray = phased.SteeringVector('SensorArray',subarray,...
    'PropagationSpeed',c);
wc = sv_subarray(fc,steer_ang_subarrays);
array.SubarraySteering = 'Custom';
pattern(array,fc,-90:90,0,'CoordinateSystem','Rectangular',...
    'Type','powerdb','PropagationSpeed',c,'Weights',wts_array,...
    'ElementWeight',conj(wc));
legend('independent subarrays')
hold off
```



See Also

`uv2azel` | `phitheta2azel`

viewArray

System object: `phased.ReplicatedSubarray`
Package: `phased`

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

`viewArray(H)` plots the geometry of the array specified in `H`.

`viewArray(H,Name,Value)` plots the geometry of the array, with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = viewArray( ___ )` returns the handles of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

`H`

Array object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Parent`

Handle to the axes along which the array geometry is displayed.

`ShowNormals`

Set this value to `true` to show the normal directions of all elements of the array. Set this value to `false` to plot the elements without showing normal directions.

Default: `false`

`ShowLocalCoordinates`

Logical flag specifying whether to show the local coordinate axes.

Default: `true`

ShowAnnotation

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Default: `true`

Orientation

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Default: `[0;0;0]`

ShowTaper

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

ShowIndex

Vector specifying the element indices to show in the figure. Each number in the vector must be an integer between 1 and the number of elements. You can also specify the value as `'All'` to show indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

ShowSubarray

Vector specifying the indices of subarrays to highlight in the figure. Each number in the vector must be an integer between 1 and the number of subarrays. You can also specify the value as `'All'` to highlight all subarrays of the array or `'None'` to suppress subarray highlighting. Highlighting uses different colors for different subarrays.

Default: `'All'`

Title

Character vector specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

Handles of array elements in figure window.

Examples

Array of Replicated Hexagonal Arrays on a Sphere

This example shows how to construct a full array by replicating subarrays.

Create a hexagonal array to use as a subarray.

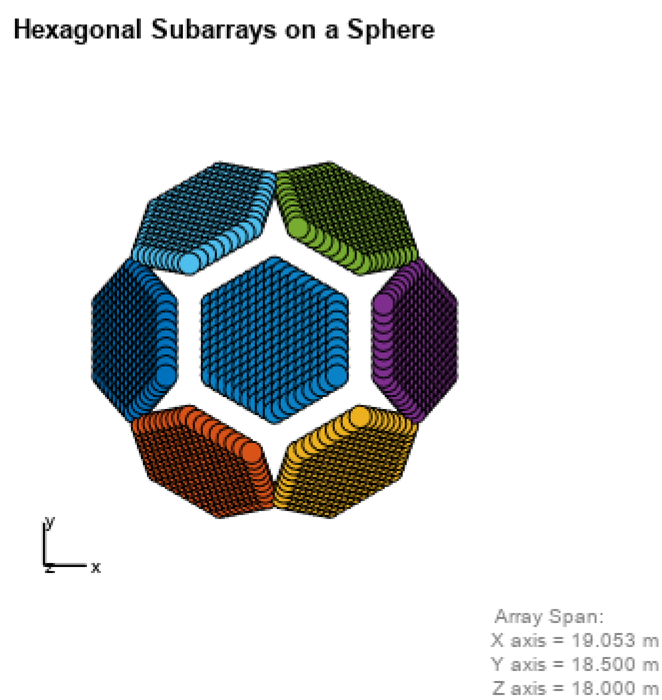
```
Nmin = 9;
Nmax = 17;
dy = 0.5;
dz = 0.5*sin(pi/3);
rowlengths = [Nmin:Nmax Nmax-1:-1:Nmin];
numels_hex = sum(rowlengths);
stopvals = cumsum(rowlengths);
startvals = stopvals-rowlengths+1;
pos = zeros(3,numels_hex);
rowidx = 0;
for m = Nmin-Nmax:Nmax-Nmin
    rowidx = rowidx+1;
    idx = startvals(rowidx):stopvals(rowidx);
    pos(2,idx) = (-(rowlengths(rowidx)-1)/2:...
        (rowlengths(rowidx)-1)/2) * dy;
    pos(3,idx) = m*dz;
end
hexa = phased.ConformalArray('ElementPosition',pos,...
    'ElementNormal',zeros(2,numels_hex));
```

Arrange copies of the hexagonal array on a sphere.

```
radius = 9;
az = [-180 -180 -180 -120 -120 -60 -60   0  0  60 60 120 120 180];
el = [-90   -30   30  -30   30 -30  30 -30 30 -30 30 -30  30  90];
numsubarrays = size(az,2);
[x,y,z] = sph2cart(deg2rad(az),deg2rad(el),...
    radius*ones(1,numsubarrays));
ha = phased.ReplicatedSubarray('Subarray',hexa,...
    'Layout','Custom',...
    'SubarrayPosition',[x; y; z], ...
    'SubarrayNormal',[az; el]);
```

Display the geometry of the array, highlighting selected subarrays with different colors.

```
viewArray(ha,'ShowSubarray',3:2:13,...
    'Title','Hexagonal Subarrays on a Sphere');
view(0,90)
```



See Also
`phased.ArrayResponse`

Topics
Phased Array Gallery

phased.RootMUSICEstimator

Package: `phased`

Root MUSIC direction of arrival (DOA) estimator for ULA and UCA

Description

The `RootMUSICEstimator` object implements the root multiple signal classification (root-MUSIC) direction of arrival estimator for uniform linear arrays (ULA) and uniform circular arrays (UCA). When a uniform circular array is used, the algorithm transforms the input to a ULA-like structure using the phase mode excitation technique [2].

To estimate the direction of arrival (DOA):

1. Define and set up your DOA estimator. See “Construction” on page 1-1487.
2. Call `step` to estimate the DOA according to the properties of `phased.RootMUSICEstimator`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.RootMUSICEstimator` creates a root MUSIC DOA estimator System object, `H`. The object estimates the signal's direction of arrival using the root MUSIC algorithm with a uniform linear array (ULA).

`H = phased.RootMUSICEstimator(Name,Value)` creates object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Sensor array System object

Sensor array specified as a System object. The sensor array must be a `phased.ULA` object or a `phased.UCA` object.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

ForwardBackwardAveraging

Perform forward-backward averaging

Set this property to `true` to use forward-backward averaging to estimate the covariance matrix for sensor arrays with conjugate symmetric array manifold.

Default: `false`

SpatialSmoothing

Spatial smoothing

The averaging number used by spatial smoothing to estimate the covariance matrix, specified as a strictly positive integer. Each additional smoothing value handles one additional coherent source, but reduces the effective number of elements by one. The maximum value of this property is *M-2*. For a ULA, *M* is the number of sensors. For a UCA, *M* is the size of the internal ULA-like array structure defined by the phase mode excitation technique. The default value of zero indicates that no spatial smoothing is employed. You can specify this property as single or double precision.

Default: `0`

NumSignalsSource

Source of number of signals

Specify the source of the number of signals as one of `'Auto'` or `'Property'`. If you set this property to `'Auto'`, the number of signals is estimated by the method specified by the `NumSignalsMethod` property.

When spatial smoothing is employed on a UCA, you cannot set the `NumSignalsSource` property to`'Auto'` to estimate the number of signals. You can use the functions `aictest` or `mdltest` independently to determine the number of signals.

Default: `'Auto'`

NumSignalsMethod

Method to estimate number of signals

Specify the method to estimate the number of signals as one of `'AIC'` or `'MDL'`. `'AIC'` uses the Akaike Information Criterion and `'MDL'` uses Minimum Description Length Criterion. This property applies when you set the `NumSignalsSource` property to `'Auto'`.

Default: `'AIC'`

`NumSignals`

Number of signals

Specify the number of signals as a positive integer scalar. This property applies when you set the `NumSignalsSource` property to `'Property'`. The number of signals must be smaller than the number of elements in the array specified in the `SensorArray` property. You can specify this property as single or double precision.

Default: 1

Methods

step	Perform DOA estimation

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Root-MUSIC Estimation of DOA for ULA

Estimate the DOA's of two signals received by a standard 10-element uniform linear array (ULA) having an element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 45 degrees in azimuth and 60 degrees in elevation.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
sULA = phased.ULA('NumElements',10,...
    'ElementSpacing',1);
sULA.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(sULA,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
sDOA = phased.RootMUSICEstimator('SensorArray',sULA,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property',...
    'NumSignals',2);
doas = step(sDOA,x + noise);
az = broadside2az(sort(doas),[20 60])
```

```
az = 1×2

   10.0001   45.0107
```

Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

[2] Mathews, C.P., Zoltowski, M.D., "Eigenstructure techniques for 2-D angle estimation with uniform circular arrays." *IEEE Transactions on Signal Processing*, vol. 42, No. 9, pp. 2395-2407, Sept. 1994.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`broadside2az` | `sensorcov` | `spsmooth` | `rootmusicdoa` | `phased.RootWSFEstimator`

step

System object: `phased.RootMUSICEstimator`
Package: `phased`

Perform DOA estimation

Syntax

```
ANG = step(H,X)
ANG = step(H,X,ElAng)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`ANG = step(H,X)` estimates the direction of arrivals (DOAs) from a signal `X` using the DOA estimator `H`. `X` is a matrix whose columns correspond to the signal channels. `ANG` is a row vector of the estimated broadside angles (in degrees). You can specify the argument `X` as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`ANG = step(H,X,ElAng)` specifies, in addition, the assumed elevation angles of the signals. This syntax is only applicable when the `SensorArray` property of the object specifies a uniform circular array (UCA). `ElAng` is a scalar between -90° and 90° and is applied to all signals. The elevation angles for all signals must be the same as required by the phase mode excitation algorithm. You can specify the argument `ElAng` as single or double precision.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Root-MUSIC Estimation of DOA for ULA

Estimate the DOA's of two signals received by a standard 10-element uniform linear array (ULA) having an element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 45 degrees in azimuth and 60 degrees in elevation.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
sULA = phased.ULA('NumElements',10,...
    'ElementSpacing',1);
sULA.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(sULA,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
sDOA = phased.RootMUSICEstimator('SensorArray',sULA,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property',...
    'NumSignals',2);
doas = step(sDOA,x + noise);
az = broadside2az(sort(doas),[20 60])
```

```
az = 1×2

   10.0001   45.0107
```

Root-MUSIC Estimation of DOA for UCA

Using the root-MUSIC algorithm, estimate the azimuth angle of arrival of two signals received by a 15-element UCA having a 1.5 meter radius. The antenna operating frequency is 150 MHz. The actual direction of arrival of the first signal is 10 degrees in azimuth and 4 degrees in elevation. The direction of arrival of the second signal is 45 degrees in azimuth and -2 degrees in elevation. In estimating the directions of arrival, assume the signals arrive from 0 degrees elevation.

Set the frequencies of the signals to 500 and 600 Hz. Set the sample rate to 8 kHz and the operating frequency to 150 MHz. Then, create the baseband signals, the UCA array and the plane wave signals.

```
fs = 8000;
fc = 150e6;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*500);
x2 = cos(2*pi*t*600);
sUCA = phased.UCA('NumElements',15,...
    'Radius',1.5);
x = collectPlaneWave(sUCA,[x1 x2],[10 4; 45 -2]',fc);
```

Add random complex Gaussian white noise to the signals.

```
rs = RandStream('mt19937ar','Seed',0);
noise = 0.1/sqrt(2)*(randn(rs,size(x))+1i*randn(rs,size(x)));
```

Create the `phased.RootMUSICEstimator` System object™.

```
sDOA = phased.RootMUSICEstimator('SensorArray',sUCA,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property',...
    'NumSignals',2);
```

Solve for the azimuth angles for zero degrees elevation.

```
elang = 0;
doas = step(sDOA, x + noise, elang);
az = sort(doas)

az = 1×2

    9.9815   44.9986
```

phased.RootWSFEstimator

Package: phased

Root WSF direction of arrival (DOA) estimator for ULA

Description

The RootWSFEstimator object implements a root weighted subspace fitting direction of arrival algorithm.

To estimate the direction of arrival (DOA):

1. Define and set up your root WSF DOA estimator. See “Construction” on page 1-1494.
2. Call step to estimate the DOA according to the properties of phased.RootWSFEstimator. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.RootWSFEstimator creates a root WSF DOA estimator System object, H. The object estimates the signal's direction of arrival using the root weighted subspace fitting (WSF) algorithm with a uniform linear array (ULA).

H = phased.RootWSFEstimator(Name,Value) creates object, H, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

SensorArray

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be a phased.ULA object.

Default: phased.ULA with default property values

PropagationSpeed

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

NumSignalsSource

Source of number of signals

Specify the source of the number of signals as one of `'Auto'` or `'Property'`. If you set this property to `'Auto'`, the number of signals is estimated by the method specified by the `NumSignalsMethod` property.

Default: `'Auto'`

NumSignalsMethod

Method to estimate number of signals

Specify the method to estimate the number of signals as one of `'AIC'` or `'MDL'`. `'AIC'` uses the Akaike Information Criterion and `'MDL'` uses the Minimum Description Length Criterion. This property applies when you set the `NumSignalsSource` property to `'Auto'`.

Default: `'AIC'`

NumSignals

Number of signals

Specify the number of signals as a positive integer scalar. This property applies when you set the `NumSignalsSource` property to `'Property'`. You can specify this property as single or double precision.

Default: `1`

Method

Iterative method

Specify the iterative method as one of `'IMODE'` or `'IQML'`.

Default: `'IMODE'`

MaximumIterationCount

Maximum number of iterations

Specify the maximum number of iterations as a positive integer scalar or `'Inf'`. This property is tunable. You can specify this property as single or double precision.

Default: `'Inf'`

Methods

step	Perform DOA estimation

Common to All System Objects	
release	Allow System object property value changes

Examples

Estimate DOA of Two Signals Using Root-WSF

First, estimate the DOAs of two signals received by a standard 10-element ULA with element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 45° in azimuth and −5° in elevation.

Create the signals with added noise. Then, create the ULA System object™.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
array.Element.FrequencyRange = [100e6 300e6];
fc = 150.0e6;
x = collectPlaneWave(array,[x1 x2],[10 20; 45 60]',fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
```

Construct WSF estimator System object.

```
estimator = phased.RootWSFEstimator('SensorArray',array, ...
    'OperatingFrequency',fc, ...
    'NumSignalsSource','Property','NumSignals',2);
```

Estimate the DOAs.

```
doas = estimator(x + noise);
doas = broadside2az(sort(doas),[20 -5])
```

```
doas =

   10.0002   20.7934
```

Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`broadside2az` | `phased.RootMUSICEstimator`

step

System object: phased.RootWSFEstimator
Package: phased

Perform DOA estimation

Syntax

ANG = step(H,X)

Description

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

ANG = step(H,X) estimates the DOAs from X using the DOA estimator H. X is a matrix whose columns correspond to channels. ANG is a row vector of the estimated broadside angles (in degrees). You can specify the argument X as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Examples

Estimate DOA of Two Signals Using Root-WSF

First, estimate the DOAs of two signals received by a standard 10-element ULA with element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10° in azimuth and 20° in elevation. The direction of the second signal is 45° in azimuth and −5° in elevation.

Create the signals with added noise. Then, create the ULA System object™.

```
fs = 8000;
t = (0:1/fs:1).';
x1 = cos(2*pi*t*300);
x2 = cos(2*pi*t*400);
array = phased.ULA('NumElements',10,'ElementSpacing',1);
```

```
array.Element.FrequencyRange = [100e6 300e6];
fc = 150.0e6;
x = collectPlaneWave(array,[x1 x2],[10 20; 45 60]',fc);
noise = 0.1*(randn(size(x)) + 1i*randn(size(x)));
```

Construct WSF estimator System object.

```
estimator = phased.RootWSFEstimator('SensorArray',array, ...
    'OperatingFrequency',fc, ...
    'NumSignalsSource','Property','NumSignals',2);
```

Estimate the DOAs.

```
doas = estimator(x + noise);
doas = broadside2az(sort(doas),[20 -5])
```

```
doas =

   10.0002   20.7934
```

phased.ScenarioViewer

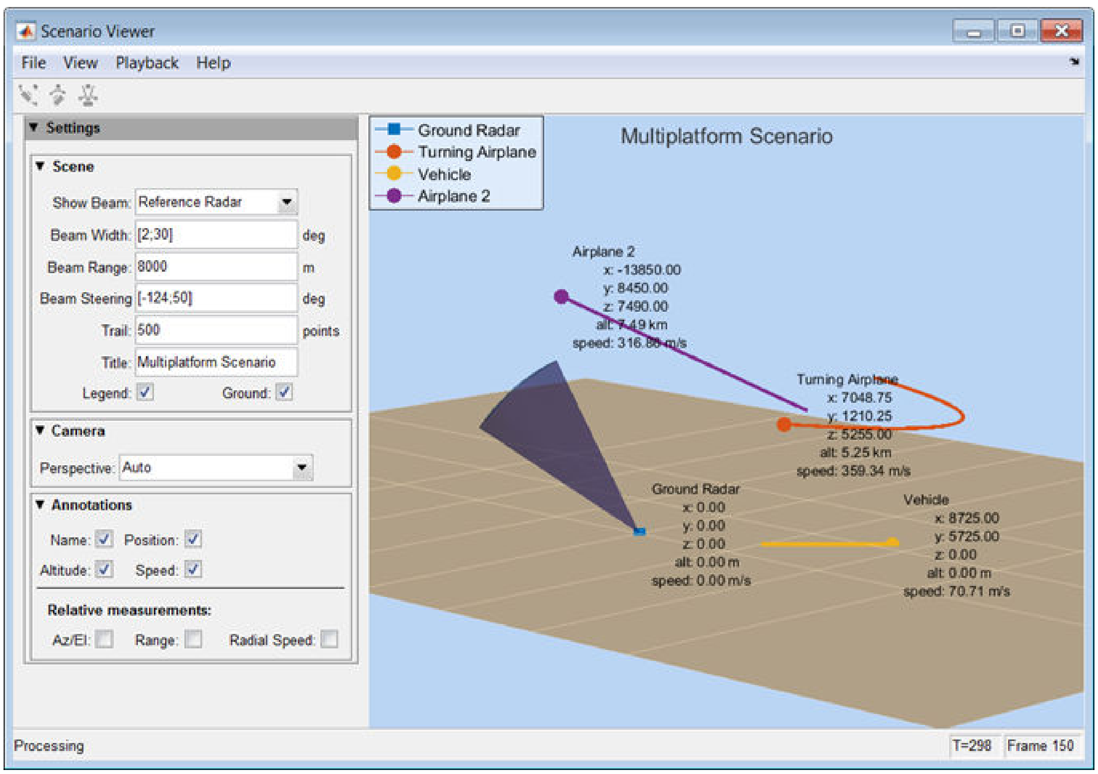
Package: `phased`

Display motion of radars and targets

Description

The `phased.ScenarioViewer` System object creates a 3-D viewer to display the motion of radars and targets that you model in your radar simulation. You can display current positions and velocities, object tracks, position and speed annotations, radar beam directions, and other object parameters. You can change radar features such as beam range and beam width during the simulation. You can use the `phased.Platform` System object to model moving objects or you can supply your own dynamic models.

This figure shows a four-object scenario consisting of a ground radar, two airplanes, and a ground vehicle. You can view the code that generated this figure in the “Visualize Multiplatform Scenario” on page 1-1509 example.



To create a scenario viewer:

1. Define and set up the `phased.ScenarioViewer` System object. See “Construction” on page 1-1501. You can set System object properties at construction time or leave them to their default values. Some properties that you set at construction time can be changed later. These properties are *tunable*.

2 Call the `step` method to update radar and target displayed positions according to the properties of the `phased.ScenarioViewer` System object. You can change tunable properties at any time.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`sIS = phased.ScenarioViewer` creates a scenario viewer System object, `sIS` having default property values.

`sIS = phased.ScenarioViewer(Name,Value)` returns a scenario viewer System object, `sIS`, with any specified property `Name` set to a specified `Value`. `Name` must appear inside single quotes (`''`). You can specify several name-value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Properties

Name — Window caption name
`'Scenario Viewer'` (default) | character vector

Window caption name, specified as a character vector. The `Name` property and the `Title` property are different.

Example: `'Multitarget Viewer'`

Data Types: `char`

ReferenceRadar — Reference radar index
`1` (default) | positive integer

Reference radar index, specified as a positive integer. This property selects one of the radars as the reference radar. Its value must be less than or equal to the number of radars that you specify in the `radar_pos` argument of the `step` method. This property is tunable. Target range, radial speed, azimuth, and elevation are defined with respect to this radar.

Example: 2

Data Types: `double`

ShowBeam — Show radar beams
`'ReferenceRadar'` (default) | `'None'` | `'All'`

Enable the display of radar beams, specified as `'ReferenceRadar'`, `'None'`, or `'All'`. This option determines which radar beams to show.

Option	Beams to show
`'ReferenceRadar'`	Show the beam of the radar specified in the `ReferenceRadar` property.
`'None'`	Do not show any radar beams.

Option	Beams to show
'All'	Show the beams for all radars.

This property is tunable.

Example: 'All'

Data Types: char

BeamWidth — Vertical and horizontal radar beam widths
15 (default) | positive, real-valued scalar | positive, real-valued 2-element column vector | positive, real-valued *N*-element row vector | positive, real-valued 2-by-*N* matrix

Vertical and horizontal radar beam widths, specified as a positive real-valued scalar, a 2-element column vector, an *N*-element row vector, or a 2-by-*N* matrix. *N* is the number of radars. All scalar, vector, and matrix entries are positive, real-valued numbers between 0–360°. Units are in degrees.

Value Specification	Interpretation
Scalar	The horizontal and vertical radar beam widths are equal and identical for all radars.
2-element column vector	The first row specifies the horizontal beam width. The second row specifies the vertical beam width. These values are identical for all radars.
N-element row vector	Each element applies to one radar. Vertical and horizontal beam widths for each radar are equal.
2-by-*N* matrix	Each column applies to one radar. The first row specifies the horizontal beam width and the second row specifies the vertical beam width for each radar.

When CameraPerspective is set to 'Radar', the System object uses this property to calculate the value of CameraViewAngle. This property is tunable.

Example: [20 10; 18 9]

Data Types: double

BeamRange — Radar beam range
1000 (default) | positive scalar | real-valued *N*- element row vector of positive values

Radar beam range, specified as a positive scalar or an *N*-element row vector, where *N* is the number of radars. Units are in meters. When specified as a scalar, all radars have the same beam range. When specified as a vector, each element corresponds to one radar. This property is tunable.

Example: [1000 1500 850]

Data Types: double

BeamSteering — Beam steering direction
[0;0] (default) | positive real-valued 2-element column vector | positive real-valued *N*-element row vector

Beam steering directions of radars, specified as a real-valued 2-element column vector of positive values or 2-by-*N* real-valued matrix of positive values. *N* is the number of radars. Beam steering angles are relative to the local coordinate axes of each radar. Units are in degrees. Each column takes the form `[azimuthangle;elevationangle]`. When only one column is specified, the beam steering directions of all radars are the same. Azimuth angles are from –180° to 180°, and the elevation angles are from –90° to 90°. This property is tunable.

Example: `[20 60 35; 5 0 10]`

Data Types: `double`

`VelocityInputPort` — Enable velocity input
`true` (default) | `false`

Enable the velocity input arguments, `radar_velocity` and `tgt_velocity`, of the `step` method, specified as `true` or `false`. Setting this property to `true` enables the input arguments. When this property is `false`, velocity vectors are estimated from the position change between consecutive updates divided by the update interval. The update interval is the inverse of the `UpdateRate` value.

Example: `false`

Data Types: `logical`

`OrientationInputPort` — Enable orientation input
`false` (default) | `true`

Enable the input of local coordinate system orientation axes, `radar_laxes` and `tgt_laxes`, to the `step` method, specified as `false` or `true`. Setting this property to `true` enables the input arguments. When this property is `false`, the orientation axes are aligned with the global coordinate axes.

Example: `true`

Data Types: `logical`

`UpdateRate` — Update rate of scenario viewer
`1` (default) | `positive scalar`

Update rate of scenario viewer, specified as a positive scalar. Units are in hertz.

Example: `2.5`

Data Types: `double`

`Title` — Display title
`''` (default) | character vector

Display title, specified as a character vector. The `Title` property and the `Name` property are different. The display title appears within the figure at the top. The name appears at the top of the figure window. This property is tunable.

Example: `'Radar and Target Display'`

Data Types: `char`

`PlatformNames` — Names of radars and targets
`'Auto'` (default) | 1-by-*(N+M)* cell array of character vectors

Names assigned to radars and targets, specified as a 1-by-*(N+M)* cell array of character vectors. *N* is the number of radars and *M* is the number of targets. Order the cell entries by radar names, followed by target names. Names appear in the legend and annotations. When you set `PlatformNames` to

`'Auto'`, names are created sequentially starting from `'Radar 1'` for radars and `'Target 1'` for targets.

Example: `{'Stationary Radar','Mobile Radar','Airplane'}`

Data Types: `cell`

`TrailLength` — Length of visible tracks
`500` (default) | positive integer | *(N+M)*-length vector of positive integers

Length of the visibility of object tracks, specified as a positive integer or *(N+M)*-length vector of positive integers. *N* is the number of radars and *M* is the number of targets. When `TrailLength` is a scalar, all tracks have the same length. When `TrailLength` is a vector, each element of the vector specifies the length of the corresponding radar or target trajectory. Order the entries by radars, followed by targets. Each call to the `step` method generates a new visible point. This property is tunable.

Example: `[100,150,100]`

Data Types: `double`

`CameraPerspective` — Camera perspective
`'Auto'` (default) | `'Custom'` | `'Radar'`

Camera perspective, specified as `'Auto'`, `'Custom'`, or `'Radar'`. When you set this property to `'Auto'`, the System object estimates appropriate values for the camera position, orientation, and view angle to show all tracks. When you set this property to `'Custom'`, you can set the camera position, orientation, and angles using camera properties or the camera toolbar. When you set this property to `'Radar'`, the System object determines the camera position, orientation, and angles from the radar position and the radar beam steering direction. This property is tunable.

Example: `'Radar'`

Data Types: `char`

`CameraPosition` — Camera position
`[x,y,z]` vector of real-values

Camera position, specified as an `[x,y,z]` vector of real values. Units are in meters. This property applies when you set `CameraPerspective` to `'Custom'`. When you do not specify this property, the System object chooses values based on your display configuration. This property is tunable.

Example: `[100,50,40]`

Data Types: `double`

`CameraOrientation` — Camera orientation
`[pan,tilt,roll]` vector of positive, real values

Camera orientation, specified as a `[pan,tilt,roll]` vector of positive, real values. Units are in degrees. Pan and roll angles take values from *–180°* to *180°*. The tilt angle takes values from *–90°* to *90°*. Camera rotations are performed in the order: pan, tilt, and roll. This property applies when you set `CameraPerspective` to `'Custom'`. When you do not specify this property, the System object chooses values based on your display configuration. This property is tunable.

Example: `[180,45,30]`

Data Types: `double`

`CameraViewAngle` — Camera view angle
real-valued scalar from 0° to 360°

Camera view angle, specified as a real-valued scalar. Units are in degrees. View angle values are in the range 0° to 360°. This property applies when you set `CameraPerspective` to `'Custom'`. When you do not specify this property, the System object chooses values based on your display configuration. This property is tunable.

Example: 75

Data Types: `double`

`ShowLegend` — Show viewer legend
`false` (default) | `true`

Option to show the viewer legend, specified as `false` or `true`. This property is tunable.

Example: `true`

Data Types: `logical`

`ShowGround` — Show ground plane of scenario
`true` (default) | `false`

Option to show the ground plane of the viewer scenario, specified as `true` or `false`. This property is tunable.

Example: `false`

Data Types: `logical`

`ShowName` — Option to annotate radar and target tracks with names
`true` (default) | `false`

Annotate radar and target tracks with names, specified as `true` or `false`. You can define custom platform names using `PlatformNames`. This property is tunable.

Example: `false`

Data Types: `logical`

`ShowPosition` — Annotate radar and target tracks with positions
`false` (default) | `true`

Option to annotate radar and target tracks with positions, specified as `false` or `true`. This property is tunable.

Example: `true`

Data Types: `logical`

`ShowRange` — Annotate radar and target tracks with ranges
`false` (default) | `true`

Option to annotate radar and target tracks with the range from the reference radar, specified as `false` or `true`. This property is tunable.

Example: `true`

Data Types: `logical`

ShowAltitude — Annotate radar and target tracks with altitude
false (default) | true

Option to annotate radar and target tracks with altitude, specified as false or true. This property is tunable.

Example: true

Data Types: logical

ShowSpeed — Annotate radar and target tracks with speed
false (default) | true

Option to annotate radar and target tracks with speed, specified as false or true. This property is tunable.

Example: true

Data Types: logical

ShowRadialSpeed — Annotate radar and target tracks with radial speed
false (default) | true

Option to annotate radar and target tracks with radial speed, specified as false or true. Radial speed is relative to the reference radar. This property is tunable.

Example: true

Data Types: logical

ShowAzEl — Annotate radar and target tracks with azimuth and elevation
false (default) | true

Option to annotate radar and target tracks with azimuth and elevation angles relative to the reference radar, specified as false or true. This property is tunable.

Example: true

Data Types: logical

Position — Viewer window size and position
[left bottom width height] vector of positive, real values

Scenario viewer window size and position, specified as a [left bottom width height] vector of positive, real values. Units are in pixels.

- left sets the position of the left edge of the window.
- bottom sets the position of the bottom edge of the window.
- width sets the width of the window.
- height sets the height of the window.

When you do not specify this property, the window is positioned at the center of the screen, with width and height taking the values 410 and 300 pixels, respectively. This property is tunable.

Example: [100,200,800,500]

Data Types: double

ReducePlotRate — Enable reduced plot rate

true (default) | false

Option to reduce the plot rate to improve performance, specified as true or false. Set this property to true to update the viewer at a reduced rate. Set this property to false to update the viewer with each call to the step method. This mode adversely affects viewer performance. This property is tunable.

Example: false

Data Types: logical

Methods

hide	Hide scenario viewer window
reset	Reset state of the System object
show	Show scenario viewer window
step	Update scenario viewer display

Common to All System Objects	
release	Allow System object property value changes

Examples

View Tracks of Stationary Radar and One Target

Visualize the tracks of a radar and a single airplane target. The radar is stationary and the airplane is moving in a straight line. Maintain the radar beam pointing at the airplane.

Create the radar and airplane platform System objects™. Set the update rate to 0.1 s.

```
updateRate = 0.1;
radarPlatform = phased.Platform(...
    'InitialPosition',[0;0;10], ...
    'Velocity',[0;0;0]);
airplanePlatforms = phased.Platform(...
    'InitialPosition',[5000.0;3500.0;6000.0],...
    'Velocity',[-300;0;0]);
```

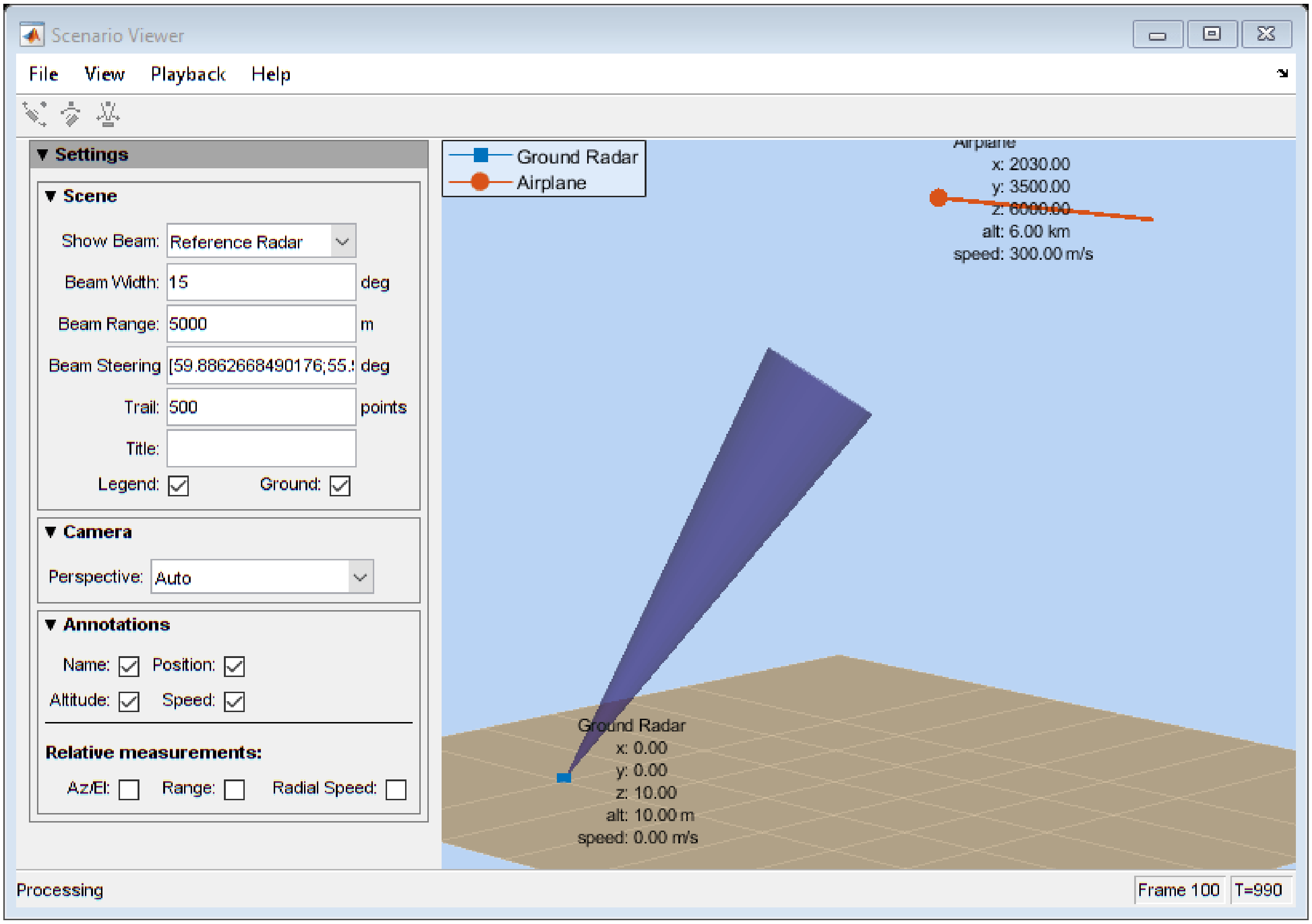
Create the phased.ScenarioViewer System object. Show the radar beam and annotate the tracks with position, speed, and altitude.

```
sSV = phased.ScenarioViewer('BeamRange',5000.0,'UpdateRate',updateRate,...
    'PlatformNames',{'Ground Radar','Airplane'},'ShowPosition',true,...
    'ShowSpeed',true,'ShowAltitude',true,'ShowLegend',true);
```

Run the scenario. At each step, compute the angle to the target. Use that angle to steer the radar beam toward the target.

```
for i = 1:100
    [radar_pos,radar_vel] = step(radarPlatform,updateRate);
    [tgt_pos,tgt_vel] = step(airplanePlatforms,updateRate);
```

```
    [rng,ang] = rangeangle(tgt_pos,radar_pos);
    sSV.BeamSteering = ang;
    step(sSV,radar_pos,radar_vel,tgt_pos,tgt_vel);
    pause(0.1);
end
```



View Tracks of Airborne Radar and Ground Target

Visualize the tracks of an airborne radar and a ground vehicle target. The airborne radar is carried by a drone flying at an altitude of 5 km.

Create the drone radar and ground vehicle using `phased.Platform` System objects™. Set the update rate to 0.1 s.

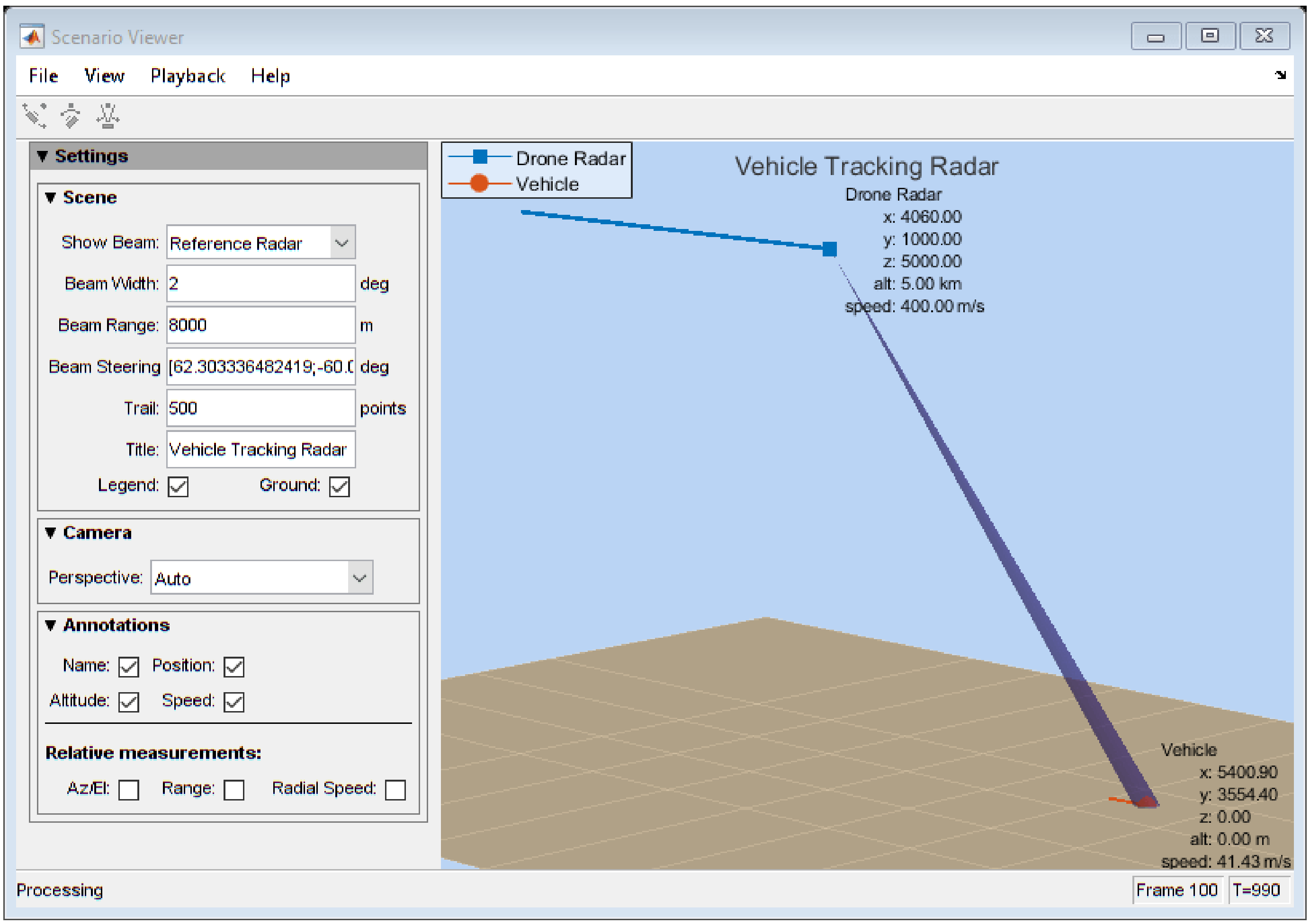
```
updateRate = 0.1;
drone = phased.Platform(...
    'InitialPosition',[100;1000;5000], ...
    'Velocity',[400;0;0]);
vehicle = phased.Platform('MotionModel','Acceleration',...
    'InitialPosition',[5000.0;3500.0;0.0],...
    'InitialVelocity',[40;5;0],'Acceleration',[0.1;0.1;0]);
```

Create the `phased.ScenarioViewer` System object. Show the radar beam and annotate the tracks with position, speed, and altitude.

```
viewer = phased.ScenarioViewer('BeamRange',8000.0,'BeamWidth',2,'UpdateRate',updateRate,...
    'PlatformNames',{'Drone Radar','Vehicle'},'ShowPosition',true,...
    'ShowSpeed',true,'ShowAltitude',true,'ShowLegend',true,'Title','Vehicle Tracking Radar');
```

Run the scenario. At each step, compute the angle to the target. Use that angle to steer the radar beam toward the target.

```
for i = 1:100
    [radar_pos,radar_vel] = step(drone,updateRate);
    [tgt_pos,tgt_vel] = step(vehicle,updateRate);
    [rng,ang] = rangeangle(tgt_pos,radar_pos);
    viewer.BeamSteering = ang;
    viewer(radar_pos,radar_vel,tgt_pos,tgt_vel)
    pause(.1)
end
```



Visualize Multiplatform Scenario

This example shows how to create and display a multiplatform scenario containing a ground-based stationary radar, a turning airplane, a constant-velocity airplane, and a moving ground vehicle. The turning airplane follows a parabolic flight path while descending at a rate of 20 m/s.

Specify the scenario refresh rate at 0.5 Hz. For 150 steps, the time duration of the scenario is 300 s.

```
updateRate = 0.5;
N = 150;
```

Set up the turning airplane using the `Acceleration` model of the `phased.Platform` System object™. Specify the initial position of the airplane by range and azimuth from the ground-based radar and its elevation. The airplane is 10 km from the radar at 60° azimuth and has an altitude of 6 km. The airplane is accelerating at 10 m/s² in the negative *x*-direction.

```
airplane1range = 10.0e3;
airplane1Azimuth = 60.0;
airplane1alt = 6.0e3;
airplane1Pos0 = [cosd(airplane1Azimuth)*airplane1range;...
    sind(airplane1Azimuth)*airplane1range;airplane1alt];
airplane1Vel0 = [400.0;-100.0;-20];
airplane1Accel = [-10.0;0.0;0.0];
airplane1platform = phased.Platform('MotionModel','Acceleration',...
    'AccelerationSource','Input port','InitialPosition',airplane1Pos0,...
    'InitialVelocity',airplane1Vel0,'OrientationAxesOutputPort',true,...
    'InitialOrientationAxes',eye(3));
```

Set up the stationary ground radar at the origin of the global coordinate system. To simulate a rotating radar, change the ground radar beam steering angle in the processing loop.

```
groundRadarPos = [0,0,0]';
groundRadarVel = [0,0,0]';
groundradarplatform = phased.Platform('MotionModel','Velocity',...
    'InitialPosition',groundRadarPos,'Velocity',groundRadarVel,...
    'InitialOrientationAxes',eye(3));
```

Set up the ground vehicle to move at a constant velocity.

```
groundVehiclePos = [5e3,2e3,0]';
groundVehicleVel = [50,50,0]';
groundvehicleplatform = phased.Platform('MotionModel','Velocity',...
    'InitialPosition',groundVehiclePos,'Velocity',groundVehicleVel,...
    'InitialOrientationAxes',eye(3));
```

Set up the second airplane to also move at constant velocity.

```
airplane2Pos = [8.5e3,1e3,6000]';
airplane2Vel = [-300,100,20]';
airplane2platform = phased.Platform('MotionModel','Velocity',...
    'InitialPosition',airplane2Pos,'Velocity',airplane2Vel,...
    'InitialOrientationAxes',eye(3));
```

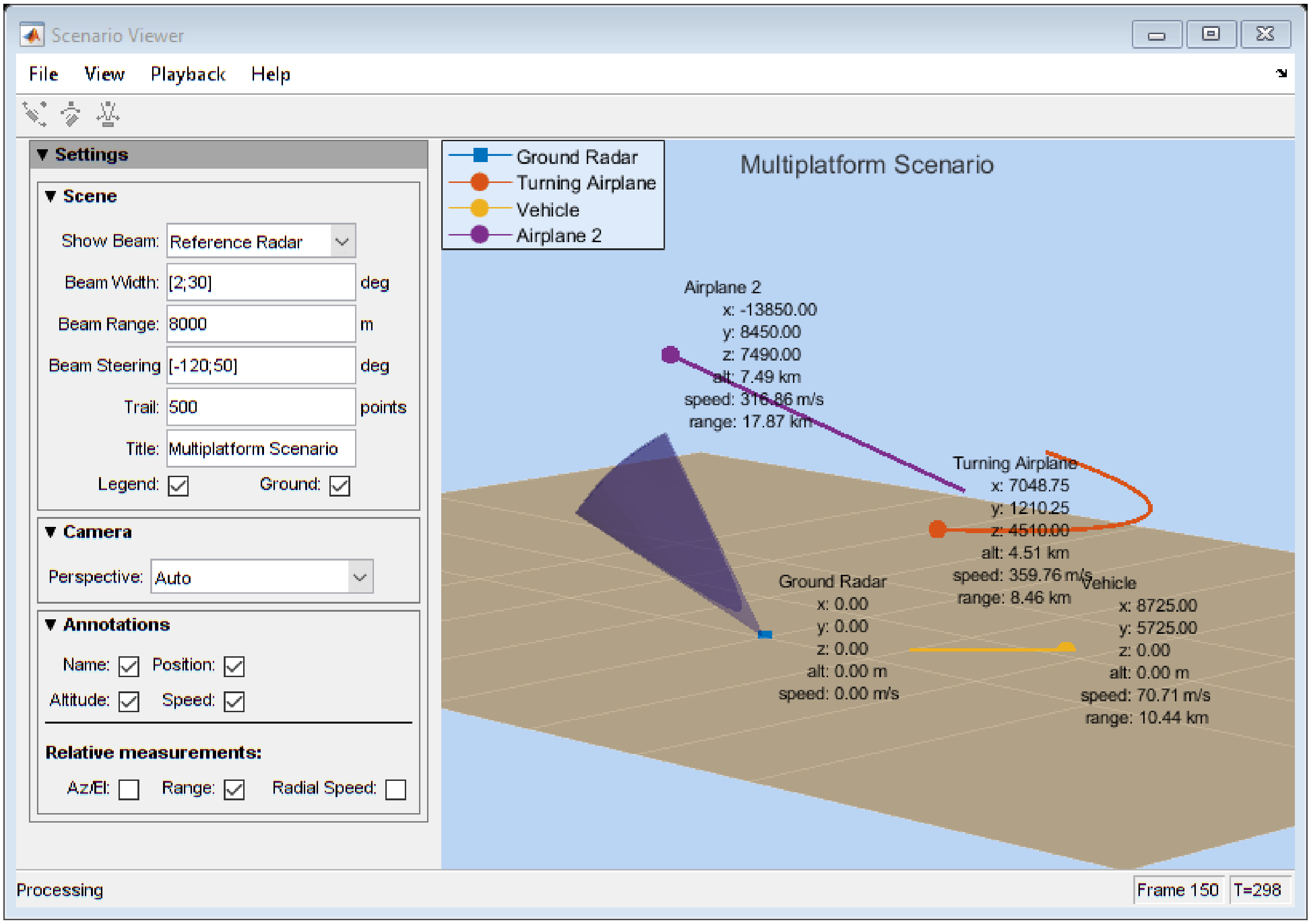
Set up the scenario viewer. Specify the radar as having a beam range of 8 km, a vertical beam width of 30°, and a horizontal beam width of 2°. Annotate the tracks with position, speed, altitude, and range.

```
BeamSteering = [0;50];
viewer = phased.ScenarioViewer('BeamRange',8.0e3,'BeamWidth',[2;30],'UpdateRate',updateRate,...
```

```
    'PlatformNames',{'Ground Radar','Turning Airplane','Vehicle','Airplane 2'},'ShowPosition',tru
    'ShowSpeed',true,'ShowAltitude',true,'ShowLegend',true,'ShowRange',true,...
    'Title','Multiplatform Scenario','BeamSteering',BeamSteering);
```

Step through the display processing loop, updating radar and target positions. Rotate the ground-based radar steering angle by four degrees at each step.

```
for n = 1:N
    [groundRadarPos,groundRadarVel] = groundradarplatform(updateRate);
    [airplane1Pos,airplane1Vel,airplane1Axes] = airplane1platform(updateRate,airplane1Accel);
    [vehiclePos,vehicleVel] = groundvehicleplatform(updateRate);
    [airplane2Pos,airplane2Vel] = airplane2platform(updateRate);
    viewer(groundRadarPos,groundRadarVel,[airplane1Pos,vehiclePos,airplane2Pos],...
        [airplane1Vel,vehicleVel,airplane2Vel]);
    BeamSteering = viewer.BeamSteering(1);
    BeamSteering = mod(BeamSteering + 4,360.0);
    if BeamSteering > 180.0
        BeamSteering = BeamSteering - 360.0;
    end
    viewer.BeamSteering(1) = BeamSteering;
    pause(0.2);
end
```



Version History

Introduced in R2016a

See Also

`phased.Platform` | `rangeangle`

Topics

"Scene Visualization for Phased Array System Simulation"

hide

System object: `phased.ScenarioViewer`
Package: `phased`

Hide scenario viewer window

Syntax

`hide(sSV)`

Description

`hide(sSV)` hides the display window of the `phased.ScenarioViewer` System object, `sSV`.

Input Arguments

`sSV` — Scenario viewer
`phased.ScenarioViewer` System object

Scenario viewer, specified as a `phased.ScenarioViewer` System object.

Example: `phased.ScenarioViewer`

Version History

Introduced in R2016a

reset

System object: `phased.ScenarioViewer`
Package: `phased`

Reset state of the System object

Syntax

`reset(sSV)`

Description

`reset(sSV)` resets the internal state of the `phased.ScenarioViewer` System object, `sSV`, to its initial value.

Input Arguments

`sSV` — Scenario viewer
`phased.ScenarioViewer` System object

Scenario viewer, specified as a `phased.ScenarioViewer` System object.

Example: `phased.ScenarioViewer`

Version History

Introduced in R2016a

show

System object: `phased.ScenarioViewer`
Package: `phased`

Show scenario viewer window

Syntax

`show(sSV)`

Description

`show(sSV)` shows the display window of the `phased.ScenarioViewer` System object, `sSV`.

Input Arguments

sSV — Scenario viewer

`phased.ScenarioViewer` System object

Scenario viewer, specified as a `phased.ScenarioViewer` System object.

Example: `phased.ScenarioViewer`

Version History

Introduced in R2016a

step

System object: `phased.ScenarioViewer`
Package: `phased`

Update scenario viewer display

Syntax

```
step(sSV,radar_pos,tgt_pos)
step(sSV,radar_pos,tgt_pos,radar_velocity,tgt_velocity)
step(sSV,radar_pos,radar_laxes,tgt_pos,tgt_laxes)
step(sSV,radar_pos,radar_velocity,radar_laxes,tgt_pos,tgt_velocity,tgt_laxes)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`step(sSV,radar_pos,tgt_pos)` updates the scenario viewer display with new radar positions, `radar_pos`, and target positions, `tgt_pos`. This syntax applies when `VelocityInputPort` and `OrientationInputPort` are set to `false`.

`step(sSV,radar_pos,tgt_pos,radar_velocity,tgt_velocity)` also specifies the radar velocity, `radar_velocity`, and target velocity, `tgt_velocity`. This syntax applies when `VelocityInputPort` is set to `true` and `OrientationInputPort` is set to `false`.

`step(sSV,radar_pos,radar_laxes,tgt_pos,tgt_laxes)` also specifies the radar orientation axes, `radar_laxes`, and the target orientation axes, `tgt_laxes`. This syntax applies when `VelocityInputPort` is set to `false` and `OrientationInputPort` is set to `true`.

`step(sSV,radar_pos,radar_velocity,radar_laxes,tgt_pos,tgt_velocity,tgt_laxes)` also specifies velocity and orientation axes when `VelocityInputPort` and `OrientationInputPort` are set to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

sSV — Scenario viewer

`phased.ScenarioViewer` System object

Scenario viewer, specified as a `phased.ScenarioViewer` System object.

Example: `phased.ScenarioViewer`

`radar_pos` — Radar positions
real-valued 3-by-*N* matrix

Radar positions, specified as a real-valued 3-by-*N* matrix. *N* is the number of radar tracks and must be equal to or greater than one. Each column has the form `[x;y;z]`. Position units are in meters.

Example: `[100,250,75;0,20,49;300,5,120]`

Data Types: `double`

`tgt_pos` — Target positions
real-valued 3-by-*M* matrix

Target positions, specified as a real-valued 3-by-*N* matrix. *M* is the number of target tracks and must be equal to or greater than one. Each column has the form `[x;y;z]`. Position units are in meters.

Example: `[200,40;10,40;305,15]`

Data Types: `double`

`radar_velocity` — Radar velocities
real-valued 3-by-*N* matrix

Radar velocities, specified as a real-valued 3-by-*N* matrix. *N* is the number of radar tracks and must be equal to or greater than one. Each column has the form `[vx;vy;vz]`. The dimensions of `radar_velocity` must match the dimensions of `radar_pos`. Velocity units are in meters per second.

Example: `[100,10,0;4,0,7;100,500,0]`

Data Types: `double`

`tgt_velocity` — Target velocities
real-valued 3-by-*M* matrix

Target velocities, specified as a real-valued 3-by-*M* matrix. *M* is the number of target tracks and must be equal to or greater than one. Each column has the form `[vx;vy;vz]`. The dimensions of `tgt_velocity` must match the dimensions of `target_position`. Velocity units are in meters per second.

Example: `[100,10,0;4,0,7;100,500,0]`

Data Types: `double`

`radar_laxes` — Radar local coordinate axes
real-valued 3-by-3-by-*N* array

Local coordinate axes of radar, specified as a real-valued 3-by-3-by-*N* array. *N* is the number of radar tracks. Each page (third index) represents a 3-by-3 orthogonal matrix that specifies the local coordinate axes of one radar. The columns are the unit vectors that form the *x*, *y*, and *z* axes of the local coordinate system. Array units are dimensionless.

Example: `[100,10,0;4,0,7;100,500,0]`

Data Types: `double`

`tgt_laxes` — Target local coordinate axes
real-valued 3-by-3-by-*M* array

Local coordinate axes of target, specified as a real-valued 3-by-3-by-*M* array. *M* is the number of target tracks. Each page (third index) represents a 3-by-3 orthogonal matrix that specifies the local coordinate axes of one radar. The columns are the unit vectors that form the *x*, *y*, and *z* axes of the local coordinate system. Array units are dimensionless.

Example: `[100,10,0;4,0,7;100,500,0]`

Data Types: `double`

Examples

View Tracks of Stationary Radar and One Target

Visualize the tracks of a radar and a single airplane target. The radar is stationary and the airplane is moving in a straight line. Maintain the radar beam pointing at the airplane.

Create the radar and airplane platform System objects™. Set the update rate to 0.1 s.

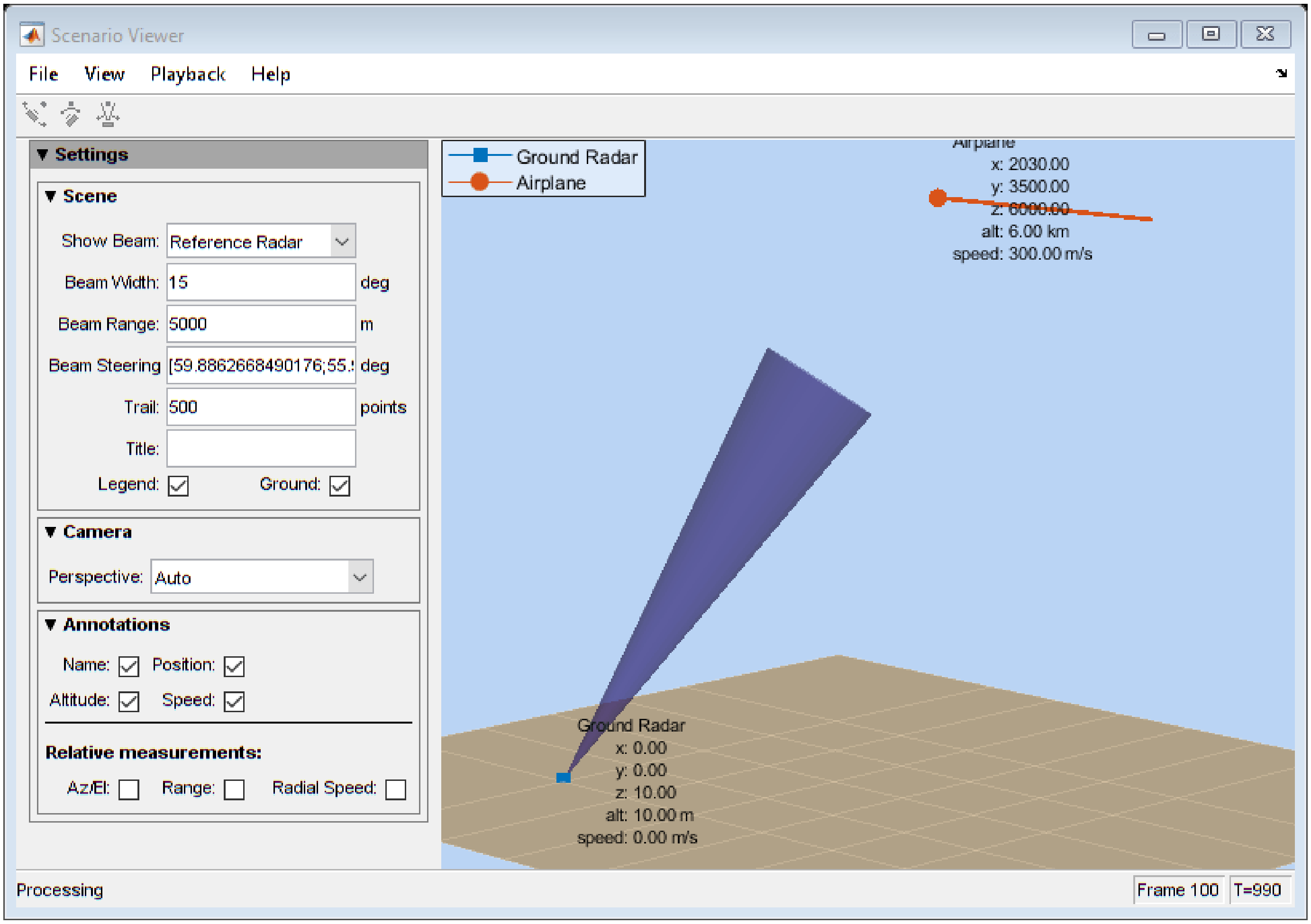
```
updateRate = 0.1;
radarPlatform = phased.Platform(...
    'InitialPosition',[0;0;10], ...
    'Velocity',[0;0;0]);
airplanePlatforms = phased.Platform(...
    'InitialPosition',[5000.0;3500.0;6000.0],...
    'Velocity',[-300;0;0]);
```

Create the `phased.ScenarioViewer` System object. Show the radar beam and annotate the tracks with position, speed, and altitude.

```
sSV = phased.ScenarioViewer('BeamRange',5000.0,'UpdateRate',updateRate,...
    'PlatformNames',{'Ground Radar','Airplane'},'ShowPosition',true,...
    'ShowSpeed',true,'ShowAltitude',true,'ShowLegend',true);
```

Run the scenario. At each step, compute the angle to the target. Use that angle to steer the radar beam toward the target.

```
for i = 1:100
    [radar_pos,radar_vel] = step(radarPlatform,updateRate);
    [tgt_pos,tgt_vel] = step(airplanePlatforms,updateRate);
    [rng,ang] = rangeangle(tgt_pos,radar_pos);
    sSV.BeamSteering = ang;
    step(sSV,radar_pos,radar_vel,tgt_pos,tgt_vel);
    pause(0.1);
end
```



View Tracks of Airborne Radar and Ground Target

Visualize the tracks of an airborne radar and a ground vehicle target. The airborne radar is carried by a drone flying at an altitude of 5 km.

Create the drone radar and ground vehicle using `phased.Platform` System objects™. Set the update rate to 0.1 s.

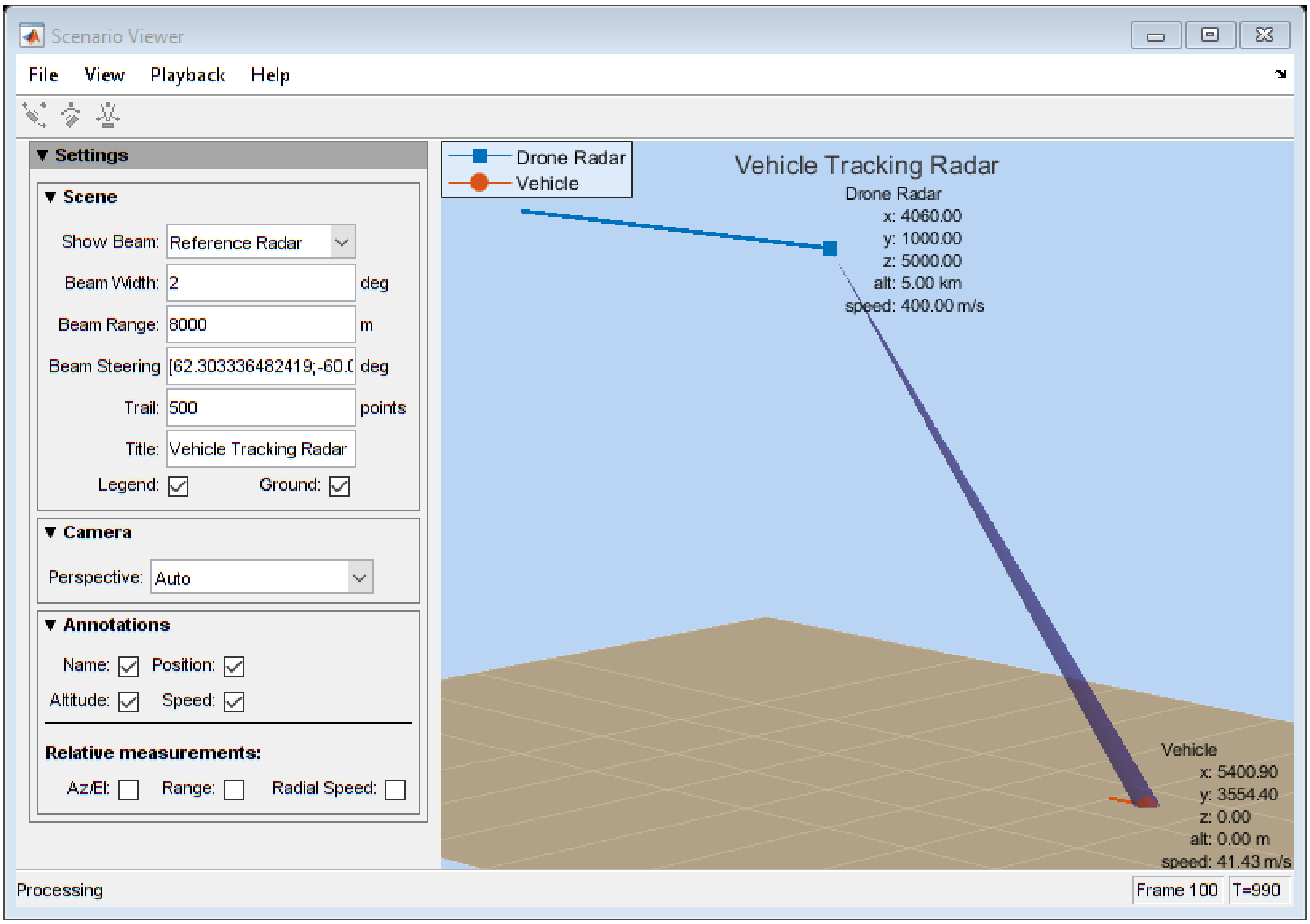
```
updateRate = 0.1;
drone = phased.Platform(...
    'InitialPosition',[100;1000;5000], ...
    'Velocity',[400;0;0]);
vehicle = phased.Platform('MotionModel','Acceleration',...
    'InitialPosition',[5000.0;3500.0;0.0],...
    'InitialVelocity',[40;5;0],'Acceleration',[0.1;0.1;0]);
```

Create the `phased.ScenarioViewer` System object. Show the radar beam and annotate the tracks with position, speed, and altitude.

```
viewer = phased.ScenarioViewer('BeamRange',8000.0,'BeamWidth',2,'UpdateRate',updateRate,...
    'PlatformNames',{'Drone Radar','Vehicle'},'ShowPosition',true,...
    'ShowSpeed',true,'ShowAltitude',true,'ShowLegend',true,'Title','Vehicle Tracking Radar');
```

Run the scenario. At each step, compute the angle to the target. Use that angle to steer the radar beam toward the target.

```
for i = 1:100
    [radar_pos,radar_vel] = step(drone,updateRate);
    [tgt_pos,tgt_vel] = step(vehicle,updateRate);
    [rng,ang] = rangeangle(tgt_pos,radar_pos);
    viewer.BeamSteering = ang;
    viewer(radar_pos,radar_vel,tgt_pos,tgt_vel)
    pause(.1)
end
```



Visualize Multiplatform Scenario

This example shows how to create and display a multiplatform scenario containing a ground-based stationary radar, a turning airplane, a constant-velocity airplane, and a moving ground vehicle. The turning airplane follows a parabolic flight path while descending at a rate of 20 m/s.

Specify the scenario refresh rate at 0.5 Hz. For 150 steps, the time duration of the scenario is 300 s.

```
updateRate = 0.5;
N = 150;
```

Set up the turning airplane using the `Acceleration` model of the `phased.Platform` System object™. Specify the initial position of the airplane by range and azimuth from the ground-based radar and its elevation. The airplane is 10 km from the radar at 60° azimuth and has an altitude of 6 km. The airplane is accelerating at 10 m/s² in the negative *x*-direction.

```
airplane1range = 10.0e3;
airplane1Azimuth = 60.0;
airplane1alt = 6.0e3;
airplane1Pos0 = [cosd(airplane1Azimuth)*airplane1range;...
    sind(airplane1Azimuth)*airplane1range;airplane1alt];
airplane1Vel0 = [400.0;-100.0;-20];
airplane1Accel = [-10.0;0.0;0.0];
airplane1platform = phased.Platform('MotionModel','Acceleration',...
    'AccelerationSource','Input port','InitialPosition',airplane1Pos0,...
    'InitialVelocity',airplane1Vel0,'OrientationAxesOutputPort',true,...
    'InitialOrientationAxes',eye(3));
```

Set up the stationary ground radar at the origin of the global coordinate system. To simulate a rotating radar, change the ground radar beam steering angle in the processing loop.

```
groundRadarPos = [0,0,0]';
groundRadarVel = [0,0,0]';
groundradarplatform = phased.Platform('MotionModel','Velocity',...
    'InitialPosition',groundRadarPos,'Velocity',groundRadarVel,...
    'InitialOrientationAxes',eye(3));
```

Set up the ground vehicle to move at a constant velocity.

```
groundVehiclePos = [5e3,2e3,0]';
groundVehicleVel = [50,50,0]';
groundvehicleplatform = phased.Platform('MotionModel','Velocity',...
    'InitialPosition',groundVehiclePos,'Velocity',groundVehicleVel,...
    'InitialOrientationAxes',eye(3));
```

Set up the second airplane to also move at constant velocity.

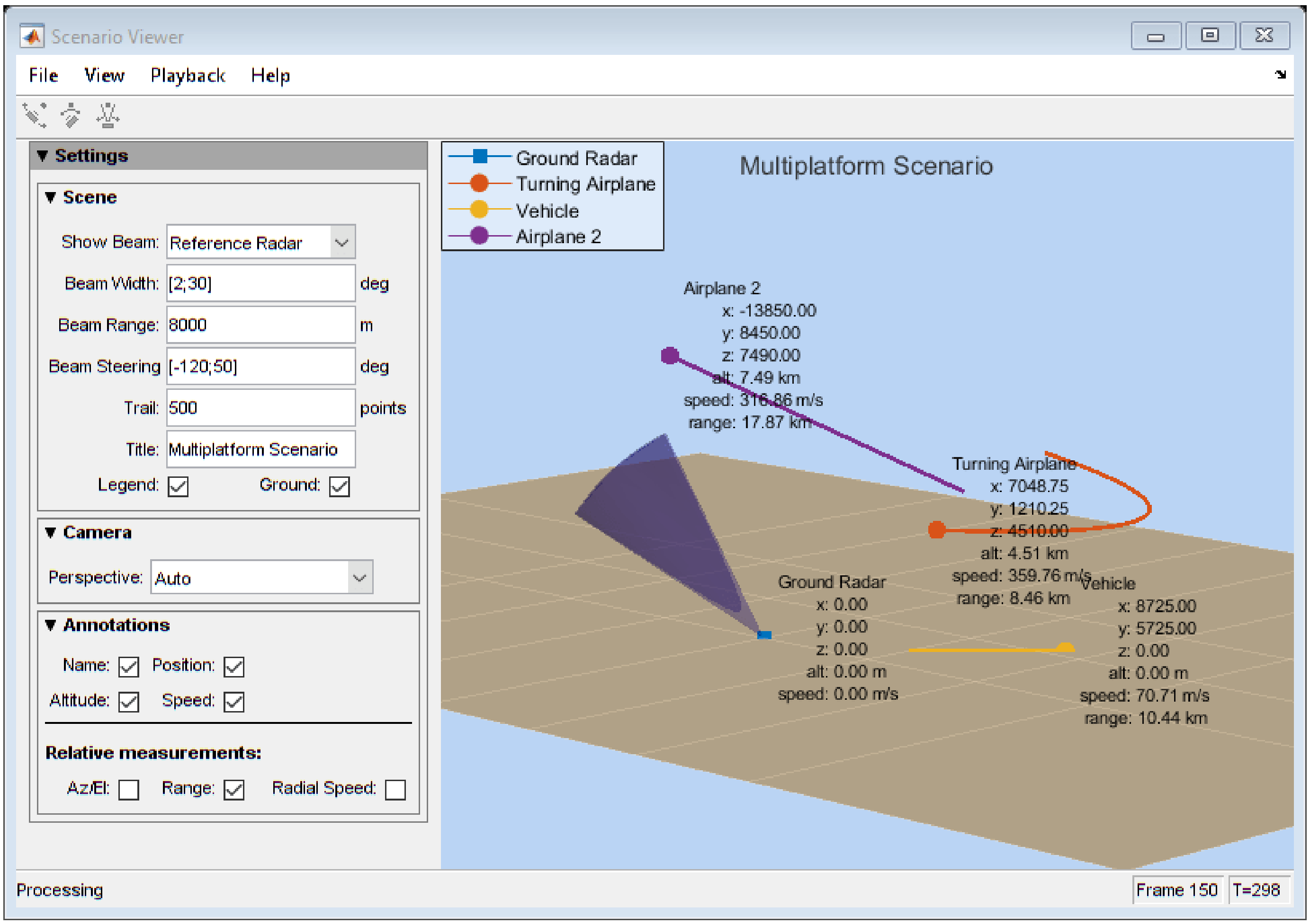
```
airplane2Pos = [8.5e3,1e3,6000]';
airplane2Vel = [-300,100,20]';
airplane2platform = phased.Platform('MotionModel','Velocity',...
    'InitialPosition',airplane2Pos,'Velocity',airplane2Vel,...
    'InitialOrientationAxes',eye(3));
```

Set up the scenario viewer. Specify the radar as having a beam range of 8 km, a vertical beam width of 30°, and a horizontal beam width of 2°. Annotate the tracks with position, speed, altitude, and range.

```
BeamSteering = [0;50];
viewer = phased.ScenarioViewer('BeamRange',8.0e3,'BeamWidth',[2;30],'UpdateRate',updateRate,...
    'PlatformNames',{'Ground Radar','Turning Airplane','Vehicle','Airplane 2'},'ShowPosition',tru
    'ShowSpeed',true,'ShowAltitude',true,'ShowLegend',true,'ShowRange',true,...
    'Title','Multiplatform Scenario','BeamSteering',BeamSteering);
```

Step through the display processing loop, updating radar and target positions. Rotate the ground-based radar steering angle by four degrees at each step.

```
for n = 1:N
    [groundRadarPos,groundRadarVel] = groundradarplatform(updateRate);
    [airplane1Pos,airplane1Vel,airplane1Axes] = airplane1platform(updateRate,airplane1Accel);
    [vehiclePos,vehicleVel] = groundvehicleplatform(updateRate);
    [airplane2Pos,airplane2Vel] = airplane2platform(updateRate);
    viewer(groundRadarPos,groundRadarVel,[airplane1Pos,vehiclePos,airplane2Pos],...
        [airplane1Vel,vehicleVel,airplane2Vel]);
    BeamSteering = viewer.BeamSteering(1);
    BeamSteering = mod(BeamSteering + 4,360.0);
    if BeamSteering > 180.0
        BeamSteering = BeamSteering - 360.0;
    end
    viewer.BeamSteering(1) = BeamSteering;
    pause(0.2);
end
```



Version History

Introduced in R2016a

phased.STAPSMIBeamformer

Package: phased

Sample matrix inversion (SMI) beamformer

Description

The SMIBeamformer object implements a sample matrix inversion space-time adaptive beamformer. The beamformer works on the space-time covariance matrix.

To compute the space-time beamformed signal:

1. Define and set up your SMI beamformer. See "Construction" on page 1-1523.
2. Call step to execute the SMI beamformer algorithm according to the properties of phased.STAPSMIBeamformer. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.STAPSMIBeamformer creates a sample matrix inversion (SMI) beamformer System object, H. The object performs the SMI space-time adaptive processing (STAP) on the input data.

H = phased.STAPSMIBeamformer(Name,Value) creates an SMI object, H, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

SensorArray

Sensor array

Sensor array specified as an array System object belonging to the phased package. A sensor array can contain subarrays.

Default: phased.ULA with default property values

PropagationSpeed

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

PRFSource

Source of pulse repetition frequency

Source of the PRF values for the STAP processor, specified as `'Property'` or `'Input port'`. When you set this property to `'Property'`, the PRF is determined by the value of the `PRF` property. When you set this property to `'Input port'`, the PRF is determined by an input argument to the `step` method at execution time.

Default: `'Property'`

PRF

Pulse repetition frequency

Pulse repetition frequency (PRF) of the received signal, specified as a positive scalar. Units are in Hertz. This property can be specified as single or double precision.

Dependencies

To enable this property, set the `PRFSource` property to `'Property'`.

Default: `1`

DirectionSource

Source of targeting direction

Specify whether the targeting direction for the STAP processor comes from the `Direction` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `Direction` property of this object specifies the targeting direction.
`'Input port'`	An input argument in each invocation of `step` specifies the targeting direction.

Default: `'Property'`

Direction

Targeting direction

Specify the targeting direction of the SMI processor as a column vector of length 2. The direction is specified in the format of `[AzimuthAngle; ElevationAngle]` (in degrees). Azimuth angle should be between –180 and 180. Elevation angle should be between –90 and 90. This property applies when you set the `DirectionSource` property to `'Property'`. You can specify this property as single or double precision.

Default: [0; 0]

NumPhaseShifterBits

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed. You can specify this property as single or double precision.

Default: 0

DopplerSource

Source of targeting Doppler

Specify whether the targeting Doppler for the STAP processor comes from the Doppler property of this object or from an input argument in step. Values of this property are:

'Property'	The Doppler property of this object specifies the Doppler.
'Input port'	An input argument in each invocation of step specifies the Doppler.

Default: 'Property'

Doppler

Targeting Doppler frequency

Specify the targeting Doppler of the STAP processor as a scalar. This property applies when you set the DopplerSource property to 'Property'. You can specify this property as single or double precision.

Default: 0

NumGuardCells

Number of guarding cells

Specify the number of guard cells used in the training as an even integer. This property specifies the total number of cells on both sides of the cell under test. You can specify this property as single or double precision.

Default: 2, indicating that there is one guard cell at both the front and back of the cell under test

NumTrainingCells

Number of training cells

Specify the number of training cells used in the training as an even integer. Whenever possible, the training cells are equally divided before and after the cell under test. You can specify this property as single or double precision.

Default: 2, indicating that there is one training cell at both the front and back of the cell under test

`WeightsOutputPort`

Output processing weights

To obtain the weights used in the STAP processor, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the weights, set this property to `false`.

Default: `false`

Methods

step	Perform SMI STAP processing on input data

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Process Data Cube Using SMI

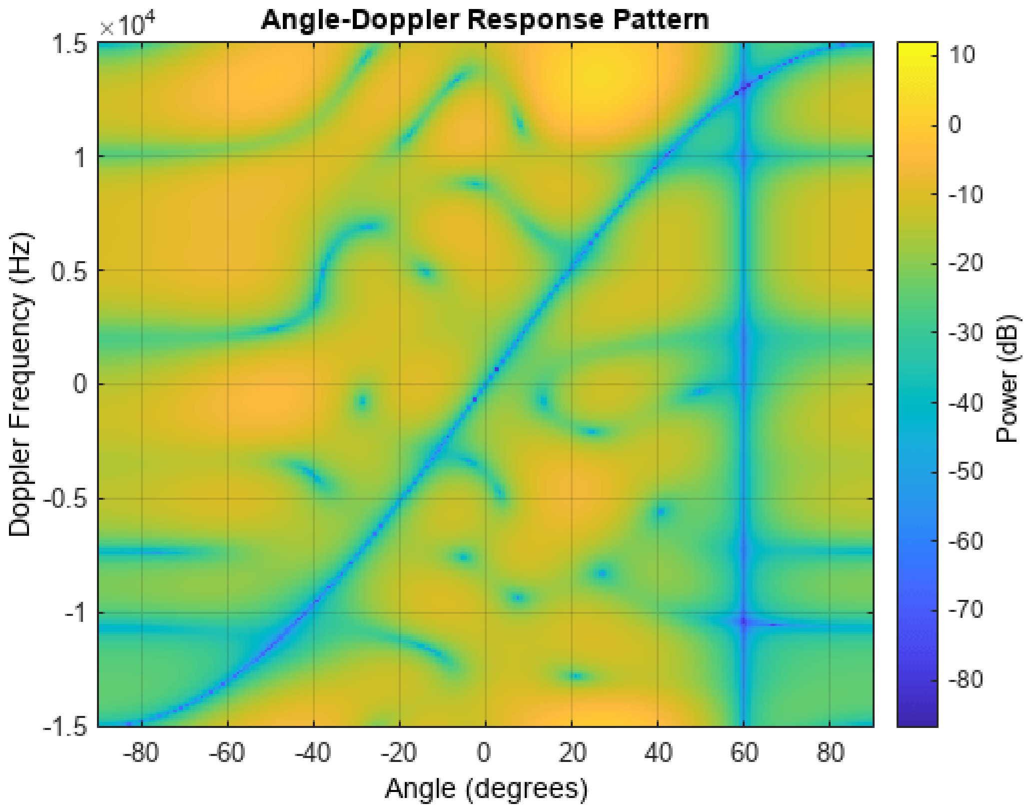
Process a data cube using an SMI beamformer. The weights are calculated for the 71st cell of a collected data cube pointing in the azimuth and elevation direction *(45°,-35°)* and with a Doppler of 12.980 kHz.

Load the cube data and create the SMI beamformer.

```
load STAPExampleData;
beamformer = phased.STAPSMIBeamformer('SensorArray',STAPEx_HArray, ...
    'PRF',STAPEx_PRF,'PropagationSpeed',STAPEx_PropagationSpeed, ...
    'OperatingFrequency',STAPEx_OperatingFrequency, ...
    'NumTrainingCells',100,'WeightsOutputPort',true, ...
    'DirectionSource','Input port','DopplerSource','Input port');
[y,w] = beamformer(STAPEx_ReceivePulse,71,[45;-35],12.980e3);
```

Plot the angle-doppler response.

```
response = phased.AngleDopplerResponse( ...
    'SensorArray',beamformer.SensorArray, ...
    'OperatingFrequency',beamformer.OperatingFrequency, ...
    'PRF',beamformer.PRF,'PropagationSpeed',beamformer.PropagationSpeed);
plotResponse(response,w)
```



Algorithms

Weight Computation

The optimum beamformer weights are

$$w = kR^{-1}v$$

where:

- k is a scalar
- R represents the space-time covariance matrix
- v indicates the space-time steering vector

Because the space-time covariance matrix is unknown, you must estimate that matrix from the data. The sample matrix inversion (SMI) algorithm estimates the covariance matrix by designating a number of range gates to be training cells. Because you use the training cells to estimate the interference covariance, these cells should not contain target returns. To prevent target returns from contaminating the estimate of the interference covariance, you can specify insertion of a number of guard cells before and after the designated target cell.

To use the general algorithm for estimating the space-time covariance matrix:

1. Assume you have a M-by-N-by-K matrix. M represents the number of slow-time samples, and N is the number of array sensors. K is the number of training cells (range gates for training). Also assume that the number of training cells is an even integer and that you can designate K/2 training cells before and after the target range gate excluding the guard cells. Reshape the M-by-N-by-K matrix into a MN-by-K matrix by letting X denote the MN-by-K matrix.
2. Estimate the space-time covariance matrix as

   $$\frac{1}{K}XX^H$$

3. Invert the space-time covariance matrix estimate.
4. Obtain the beamforming weights by multiplying the sample space-time covariance matrix inverse by the space-time steering vector.

Single Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Guerci, J. R. *Space-Time Adaptive Processing for Radar*. Boston: Artech House, 2003.

[2] Ward, J. “Space-Time Adaptive Processing for Airborne Radar Data Systems,” *Technical Report 1015*, MIT Lincoln Laboratory, December, 1994.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See Also

`phased.ADPCACanceller` | `phased.AngleDopplerResponse` | `phased.DPCACanceller` | `uv2azel` | `phitheta2azel`

step

System object: `phased.STAPSMIBeamformer`
Package: `phased`

Perform SMI STAP processing on input data

Syntax

```
Y = step(H,X,CUTIDX)
Y = step(H,X,CUTIDX,PRF)
Y = step(H,X,CUTIDX,ANG)
Y = step(H,X,CUTIDX,DOP)
[Y,W] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X,CUTIDX)` applies SMI processing to the input data, `X`. `X` must be a 3-dimensional *M*-by-*N*-by-*P* numeric array whose dimensions are (range, channels, pulses). The processing weights are calculated according to the range cell specified by `CUTIDX`. The targeting direction and the targeting Doppler are specified by `Direction` and `Doppler` properties, respectively. `Y` is a column vector of length *M*. This syntax is available when the `DirectionSource` property is `'Property'` and the `DopplerSource` property is `'Property'`.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`Y = step(H,X,CUTIDX,PRF)` uses `PRF` as the pulse repetition frequency. This syntax is available when the `PRFSource` property is `'Input port'`.

`Y = step(H,X,CUTIDX,ANG)` uses `ANG` as the targeting direction. This syntax is available when the `DirectionSource` property is `'Input port'`. `ANG` must be a 2-by-1 vector in the form of `[AzimuthAngle; ElevationAngle]` (in degrees). The azimuth angle must be between –180 and 180. The elevation angle must be between –90 and 90.

`Y = step(H,X,CUTIDX,DOP)` uses `DOP` as the targeting Doppler frequency (in hertz). This syntax is available when the `DopplerSource` property is `'Input port'`. `DOP` must be a scalar.

You can combine optional input arguments when their enabling properties are set: `Y = step(H,X,CUTIDX,ANG,DOP)`

`[Y,W] = step( ___ )` returns the additional output, `W`, as the processing weights. This syntax is available when the `WeightsOutputPort` property is `true`. `W` is a column vector of length *N*P*.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Process Data Cube Using SMI

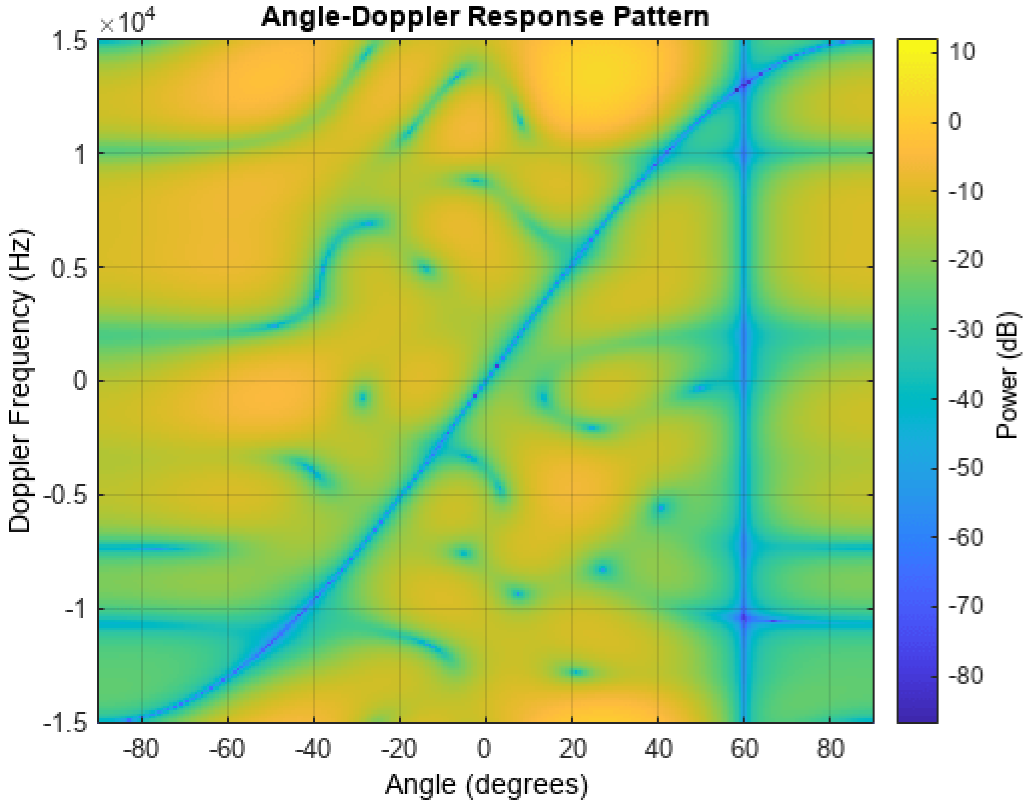
Process a data cube using an SMI beamformer. The weights are calculated for the 71st cell of a collected data cube pointing in the azimuth and elevation direction *(45°,-35°)* and with a Doppler of 12.980 kHz.

Load the cube data and create the SMI beamformer.

```
load STAPExampleData;
beamformer = phased.STAPSMIBeamformer('SensorArray',STAPEx_HArray, ...
    'PRF',STAPEx_PRF,'PropagationSpeed',STAPEx_PropagationSpeed, ...
    'OperatingFrequency',STAPEx_OperatingFrequency, ...
    'NumTrainingCells',100,'WeightsOutputPort',true, ...
    'DirectionSource','Input port','DopplerSource','Input port');
[y,w] = beamformer(STAPEx_ReceivePulse,71,[45;-35],12.980e3);
```

Plot the angle-doppler response.

```
response = phased.AngleDopplerResponse( ...
    'SensorArray',beamformer.SensorArray, ...
    'OperatingFrequency',beamformer.OperatingFrequency, ...
    'PRF',beamformer.PRF,'PropagationSpeed',beamformer.PropagationSpeed);
plotResponse(response,w)
```



See Also

`uv2azel` | `phitheta2azel`

phased.ShortDipoleAntennaElement

Package: `phased`

Short-dipole antenna element

Description

The `phased.ShortDipoleAntennaElement` object models a short-dipole antenna element. A short-dipole antenna is a center-fed wire whose length is much shorter than one wavelength. This antenna object only supports polarized fields.

To compute the response of the antenna element for specified directions:

1. Create the `phased.ShortDipoleAntennaElement` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
antenna = phased.ShortDipoleAntennaElement
antenna = phased.ShortDipoleAntennaElement(Name,Value)
RESP = antenna(H,FREQ,ANG)
```

Description

`antenna = phased.ShortDipoleAntennaElement` creates the system object, `h`, to model a short-dipole antenna element.

`antenna = phased.ShortDipoleAntennaElement(Name,Value)` creates the system object, `antenna`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`FrequencyRange` — Antenna operating frequency range
`[0 1e20]` (default) | real-valued 1-by-2 row vector

Antenna operating frequency range specified as a 1-by-2 row vector in the form of [LowerBound HigherBound]. This vector defines the frequency range over which the antenna has a response. The antenna element has zero response outside this specified frequency range.

Data Types: double

AxisDirection — Dipole axis direction
'Z' (default) | 'Y' | 'Z' | 'Custom'

Dipole axis direction, specified as one of 'X', 'Y', 'Z', or 'Custom'. The dipole axis defines the direction of the dipole current with respect to the local coordinate system. 'X' specifies a dipole along the *x*-axis, 'Y' specifies a dipole along the *y*-axis, and 'Z' specifies a dipole along the *z*-axis. An *x*-axis or *y*-axis direction is equivalent to a horizontal dipole and a *z*-axis direction is equivalent to a vertical dipole. When you set the AxisDirection property to 'Custom', you can specify the dipole axis using the CustomAxisDirection property.

Data Types: char

CustomAxisDirection — Custom dipole axis direction
[0;0;1] (default) | real-valued 3-element column vector

Custom axis direction of the dipole antenna, specified as a real-valued 3-element column vector. Each entry in the vector represents the component of the dipole axis along the *x*, *y*, and *z* axes in the local coordinate system.

Dependencies

To enable this property, set the AxisDirection property to 'Custom'.

Data Types: double

Usage

Syntax

RESP = antenna(H,FREQ,ANG)

Description

RESP = antenna(H,FREQ,ANG) returns the antenna's voltage response, RESP, at the operating frequencies specified in FREQ and in the directions specified in ANG. For the short-dipole antenna element object, RESP is a MATLAB struct containing two fields, RESP.H and RESP.V, representing the horizontal and vertical polarization components of the antenna's response. Each field is an *M*-by-*L* matrix containing the antenna response at the *M* angles specified in ANG and at the *L* frequencies specified in FREQ.

Input Arguments

FREQ — Operating frequency of antenna element
nonnegative scalar | nonnegative, real-valued 1-by-*L* row vector

Operating frequency of the antenna element, specified as a nonnegative scalar or nonnegative, real-valued 1-by-*L* row vector. Frequency units are in Hz.

FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element produces no response and the response is returned as –Inf. Element objects use the FrequencyRange property, except for phased.CustomAntennaElement, which uses the FrequencyVector property.

Example: [1e8 2e6]

Data Types: double

ANG — Azimuth and elevation angles of response directions
real-valued 1-by-*M* row vector | real-valued 2-by-*M* matrix

Azimuth and elevation angles of the response directions, specified as a real-valued 1-by-*M* row vector or a real-valued 2-by-*M* matrix, where *M* is the number of angular directions. Angle units are in degrees. The azimuth angle must lie in the range –180° to 180°, inclusive. The elevation angle must lie in the range –90° to 90°, inclusive.

- If ANG is a 1-by-*M* vector, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be zero.
- If ANG is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form [azimuth;elevation].

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy*-plane. This angle is positive when measured toward the *z*-axis. See the definition of "Azimuth and Elevation Angles".

Example: [110 125; 15 10]

Data Types: double

Output Arguments

RESP — Antenna voltage response
struct

Voltage response of antenna element returned as a MATLAB struct with fields RESP.H and RESP.V. Both RESP.H and RESP.V contain responses for the horizontal and vertical polarization components of the antenna radiation pattern. Both RESP.H and RESP.V are *M*-by-*L* matrices. In these matrices, *M* represents the number of angles specified in ANG, and *L* represents the number of frequencies specified in FREQ.

Data Types: double

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named obj, use this syntax:

```
release(obj)
```

Specific to Antenna and Transducer Element System Objects

beamwidth	Compute and display beamwidth of sensor element pattern
directivity	Directivity of antenna or transducer element
isPolarizationCapable	Antenna element polarization capability
pattern	Plot antenna or transducer element directivity and patterns

patternAzimuth	Plot antenna or transducer element directivity and pattern versus azimuth
patternElevation	Plot antenna or transducer element directivity and pattern versus elevation

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Response of Short-Dipole Antenna

Find the response of a short-dipole antenna element at boresight, (0°,0°), and off boresight, (30°,0°). The antenna operates at 256 MHz.

```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Y');
ang = [0 30;0 0];
fc = 250e6;
resp = antenna(fc,ang)
```

```
resp = struct with fields:
    H: [2x1 double]
    V: [2x1 double]
```

Horizontal response.

```
disp(resp.H)
```

```
   -1.2247
   -1.0607
```

Vertical response.

```
disp(resp.V)
```

```
     0
     0
```

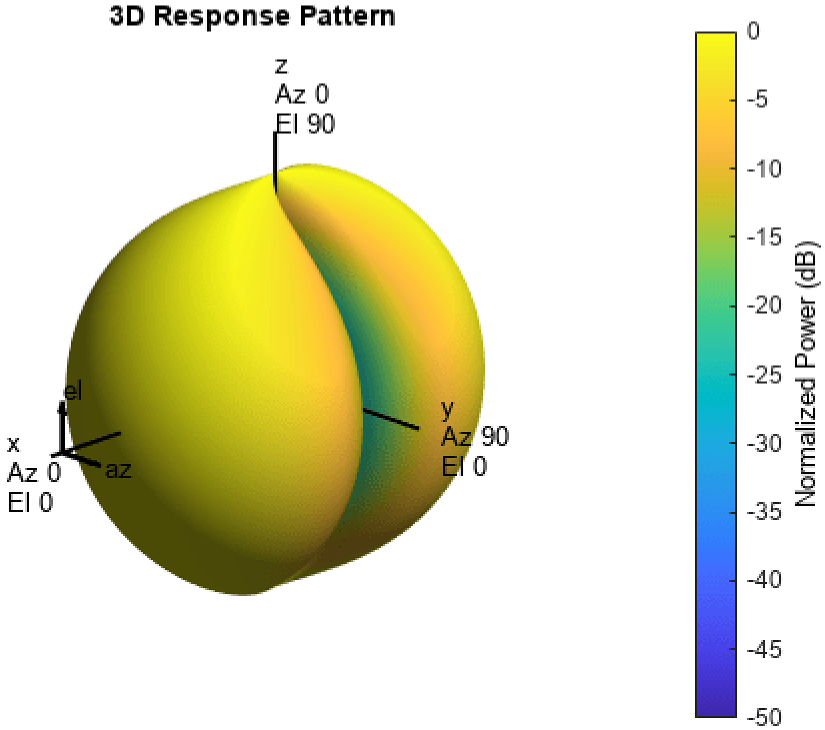
Short-Dipole Antenna Aligned Along the Y-Axis

Specify a short-dipole antenna with the dipole oriented along the *y*-axis and operating at 250 MHz. Then, plot the 3-D responses for both the horizontal and vertical polarizations.

```
antenna = phased.ShortDipoleAntennaElement( ...
    'FrequencyRange',[100e6,600e6],'AxisDirection','Y');
fc = 250.0e6;
```

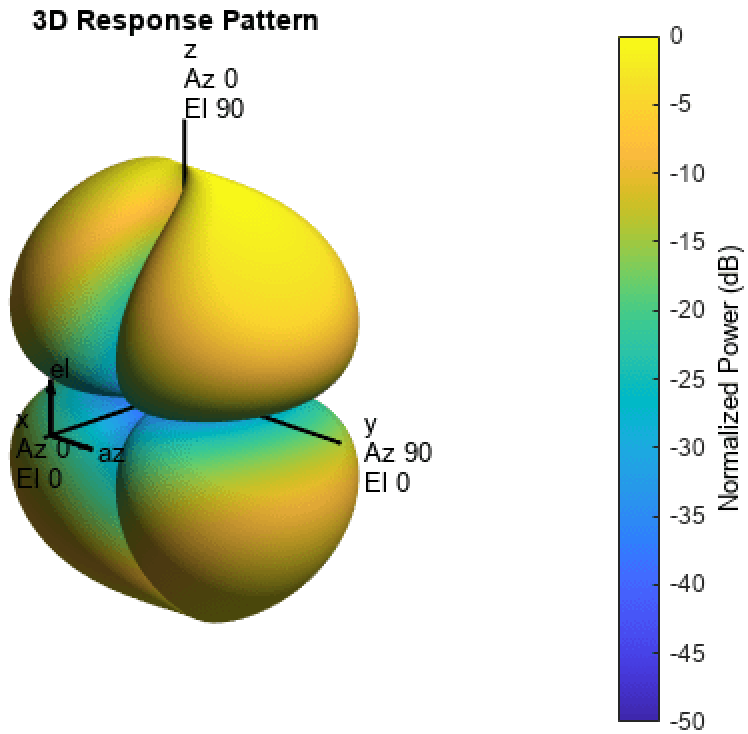
Plot the horizontal polarization response.

```
pattern(antenna,fc,-180:180,[-90:90],'CoordinateSystem','polar', ...
    'Type','powerdb','Polarization','H');
```



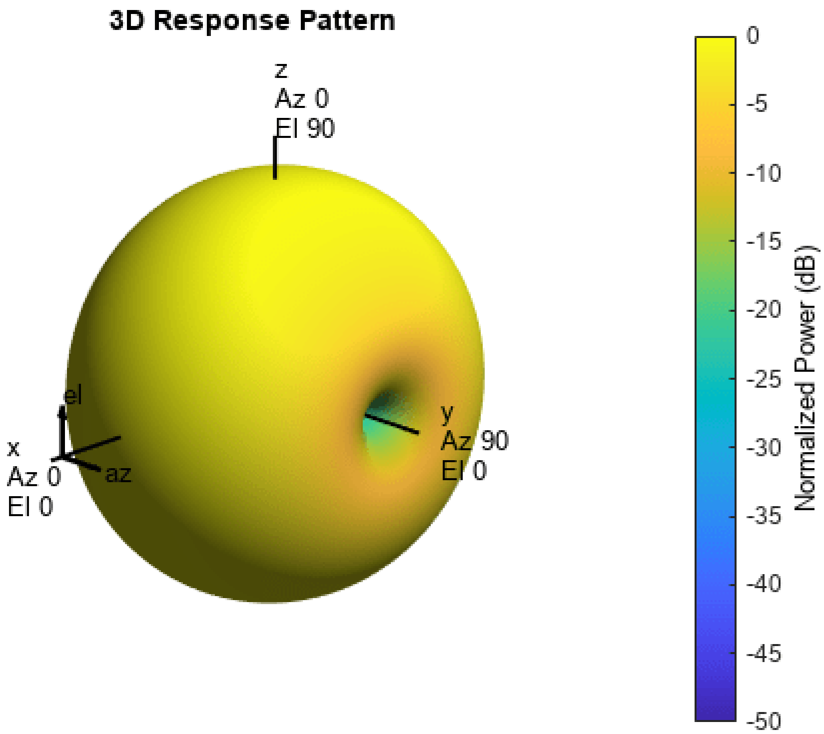
Plot the vertical polarization response.

```
pattern(antenna,fc,-180:180,[-90:90],'CoordinateSystem','polar', ...
    'Type','powerdb','Polarization','V');
```



Plot the combined response.

```
pattern(antenna,fc,-180:180,[-90:90],'CoordinateSystem','polar',...
    'Type','powerdb','Polarization','C');
```



Short-Dipole Antenna Aligned Along Arbitrary Axis

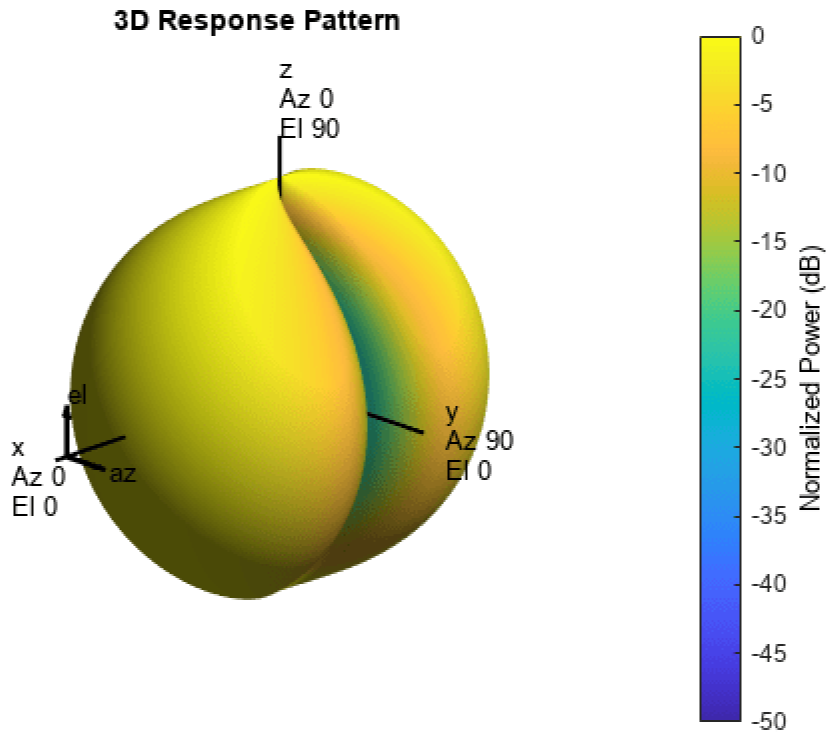
Specify a short-dipole antenna with the dipole oriented along a custom axis and operating at 250 MHz. Then, plot the 3-D responses for both the horizontal and vertical polarizations.

Create the short-dipole antenna element System object™. An easy way to create a custom axis is to rotate a unit vector using rotation functions.

```
v = rotx(30)*rotz(45)*[0;0;1];
antenna = phased.ShortDipoleAntennaElement( ...
    'FrequencyRange',[100e6,600e6],'AxisDirection','Custom', ...
    'CustomAxisDirection',v);
```

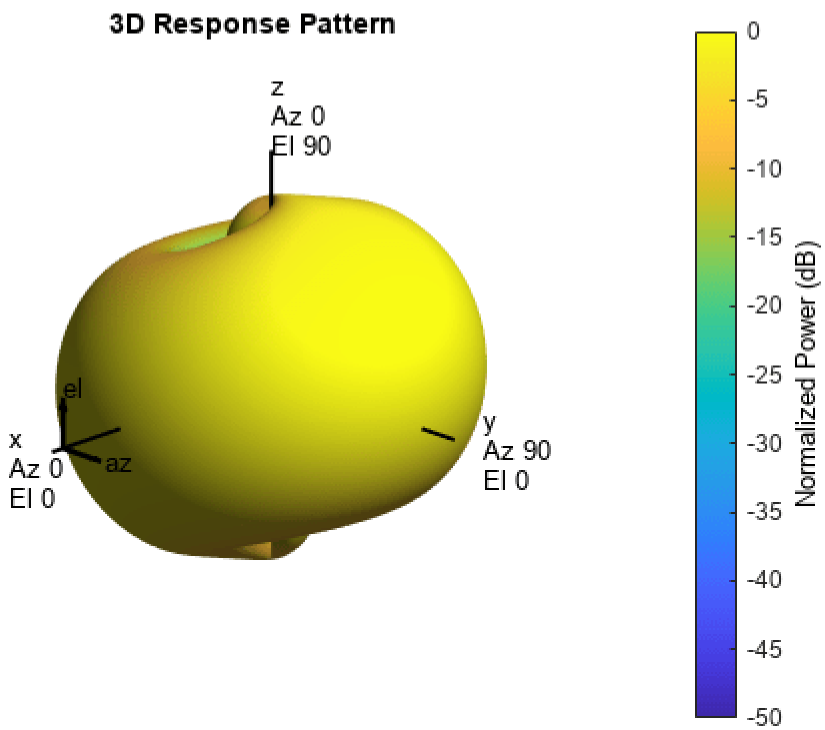
Plot the horizontal polarization response.

```
fc = 250.0e6;
pattern(antenna,fc,-180:180,[-90:90],'CoordinateSystem','polar', ...
    'Type','powerdb','Polarization','H');
```



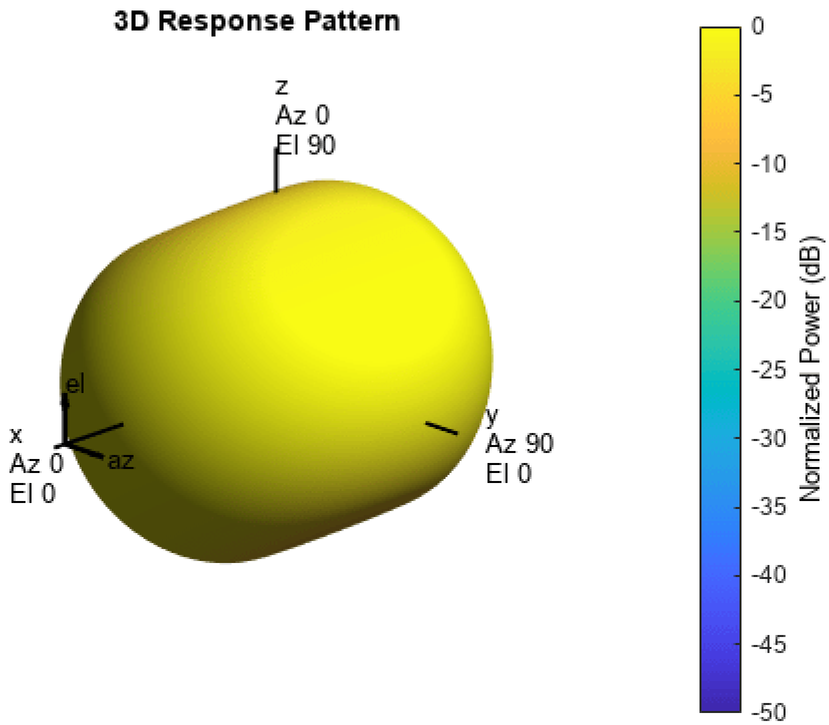
Plot the vertical polarization response.

```
pattern(antenna,fc,-180:180,[-90:90],'CoordinateSystem','polar', ...
    'Type','powerdb','Polarization','V');
```



Plot the combined response.

```
pattern(antenna,fc,-180:180,[-90:90],'CoordinateSystem','polar', ...
    'Type','powerdb','Polarization','C');
```



Algorithms

The total response of a short-dipole antenna element is a combination of its frequency response and spatial response. This System object calculates both responses using nearest neighbor interpolation and then multiplies the responses to form the total response.

Version History

Introduced in R2013a

References

[1] Mott, H., *Antennas for Radar and Communications*, John Wiley & Sons, 1992.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, and `plotResponse` methods are not supported.

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also
`phased.CosineAntennaElement` | `phased.CrossedDipoleAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.URA` | `phased.ConformalArray` | `uv2azelpat` | `phitheta2azelpat` | `uv2azel` | `phitheta2azel`

directivity

System object: phased.ShortDipoleAntennaElement
Package: phased

Directivity of short-dipole antenna element

Syntax

```
D = directivity(H,FREQ,ANGLE)
```

Description

D = directivity(H,FREQ,ANGLE) returns the "Directivity (dBi)" on page 1-1545 of a short-dipole antenna element, H, at frequencies specified by FREQ and in direction angles specified by ANGLE.

Input Arguments

H — Short-dipole antenna element

System object

Short-dipole antenna element specified as a phased.ShortDipoleAntennaElement System object.

Example: H = phased.ShortDipoleAntennaElement;

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.
- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: [1e8 2e6]

Data Types: double

ANGLE — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If ANGLE is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, [az;el]. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Short-Dipole Antenna Element

Compute the directivity of a z-directed short-dipole antenna element as a function of elevation.

Create the crossed-dipole antenna element System object™.

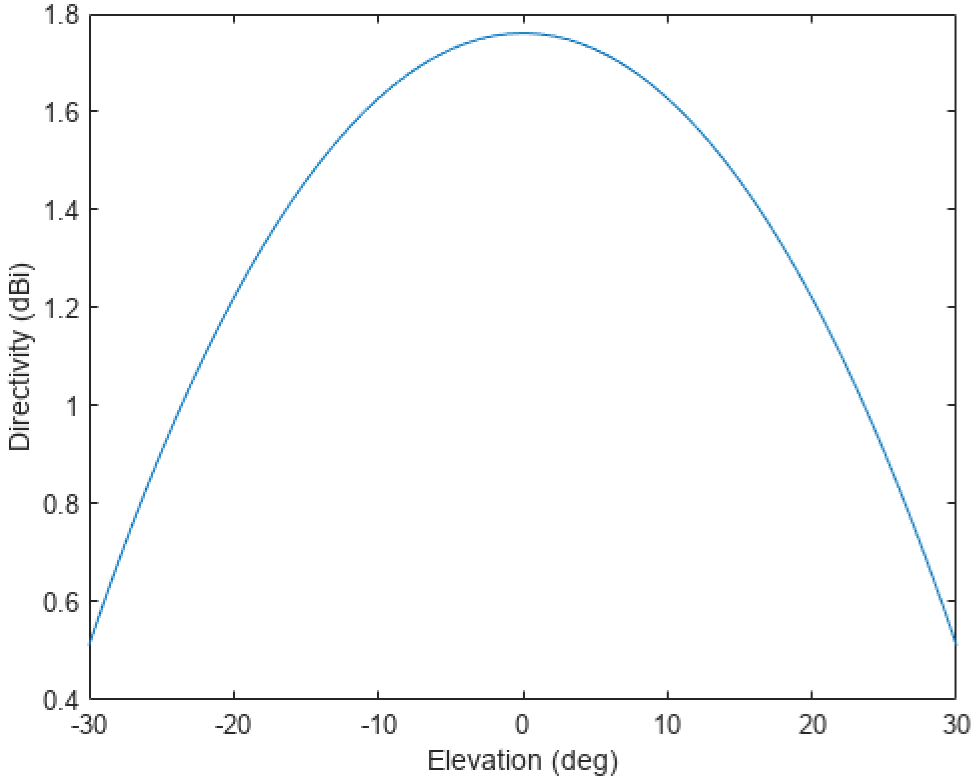
```
myAnt = phased.ShortDipoleAntennaElement;
myAnt.AxisDirection = 'Z';
myAnt.FrequencyRange = [0,10e9];
```

Select the desired angles of interest to be at constant azimuth angle at zero degrees. Set the elevation angles to center around boresight (zero degrees azimuth and zero degrees elevation). Set the frequency to 1 GHz.

```
elev = [-30:30];
azm = zeros(size(elev));
ang = [azm;elev];
freq = 1e9;
```

Plot the directivity along the constant azimuth cut.

```
d = directivity(myAnt,freq,ang);
plot(elev,d)
xlabel('Elevation (deg)');
ylabel('Directivity (dBi)');
```



More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternElevation` | `patternAzimuth`

isPolarizationCapable

System object: phased.ShortDipoleAntennaElement
Package: phased

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.ShortDipoleAntennaElement` antenna element supports polarization or not. An antenna element supports polarization if it can create or respond to polarized fields. The `phased.ShortDipoleAntennaElement` object always supports polarization.

Input Arguments

h — Short-dipole antenna element

Short-dipole antenna element specified as a `phased.ShortDipoleAntennaElement` System object.

Output Arguments

flag — Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not. Because the short-dipole antenna element supports polarization, the returned value is always `true`.

Examples

Short-Dipole Antenna Supports Polarization

Show that a `phased.ShortDipoleAntennaElement` antenna supports polarization.

```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
isPolarizationCapable(antenna)
```

```
ans = logical
   1
```

The returned value of 1 shows that this antenna supports polarization.

pattern

System object: `phased.ShortDipoleAntennaElement`
Package: `phased`

Plot short-dipole antenna element directivity and patterns

Syntax

```
pattern(sElem,FREQ)
pattern(sElem,FREQ,AZ)
pattern(sElem,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sElem,FREQ)` plots the 3-D array directivity pattern (in dBi) for the element specified in `sElem`. The operating frequency is specified in `FREQ`.

`pattern(sElem,FREQ,AZ)` plots the element directivity pattern at the specified azimuth angle.

`pattern(sElem,FREQ,AZ,EL)` plots the element directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the element pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See "Convert plotResponse to pattern" on page 1-1554 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sElem` — Short-dipole antenna element

System object

Short-dipole antenna element, specified as a `phased.ShortDipoleAntennaElement` System object.

Example: `sElem = phased.ShortDipoleAntennaElement;`

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the

pattern azimuth and elevation, respectively. AZ values must lie between –180° and 180°. EL values must lie between –90° and 90°. If 'CoordinateSystem' is set to 'uv', AZ and EL then specify *U* and *V* coordinates, respectively. AZ and EL must lie between -1 and 1.

Example: 'uv'

Data Types: char

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

PlotStyle — Plotting style
'overlay' (default) | 'waterfall'

Plotting style, specified as the comma-separated pair consisting of 'Plotstyle' and either 'overlay' or 'waterfall'. This parameter applies when you specify multiple frequencies in FREQ in 2-D plots. You can draw 2-D plots by setting one of the arguments AZ or EL to a scalar.

Data Types: char

Polarization — Polarized field component
'combined' (default) | 'H' | 'V'

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and 'combined', 'H', or 'V'. This parameter applies only when the sensors are polarization-capable and when the 'Type' parameter is not set to 'directivity'. This table shows the meaning of the display options.

'Polarization'	Display
'combined'	Combined *H* and *V* polarization components
'H'	*H* polarization component
'V'	*V* polarization component

Example: 'V'

Data Types: char

Output Arguments

PAT — Element pattern
N-by-*M* real-valued matrix

Element pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of PAT correspond to the azimuth angles in the vector specified by EL_ANG. The columns correspond to the elevation angles in the vector specified by AZ_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

Examples

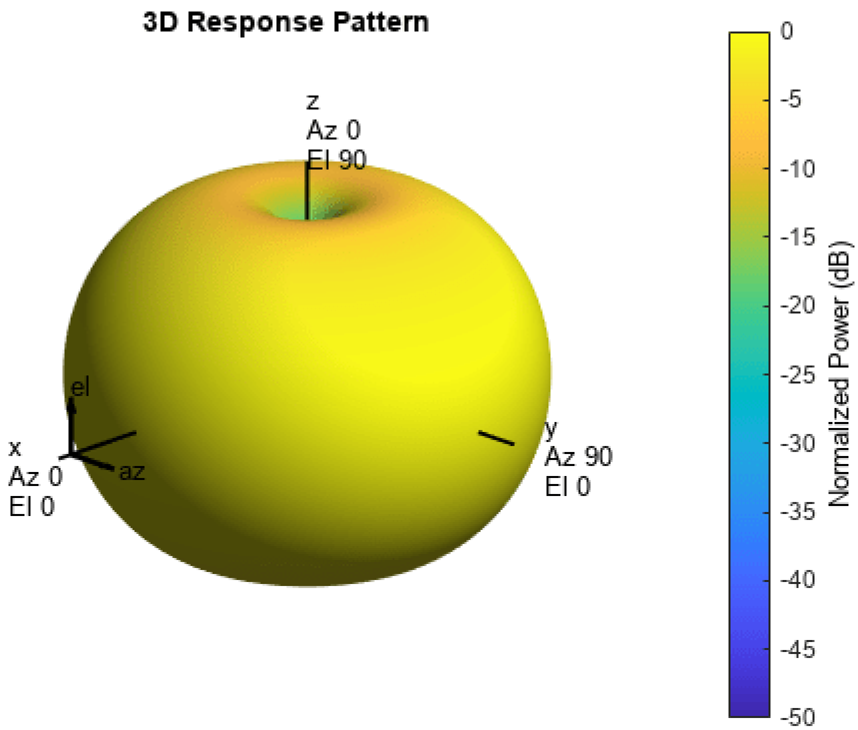
Pattern of Short-Dipole Antenna Oriented Along the Z-Axis

Specify a short-dipole antenna element with its dipole axis pointing along the z-axis. To do so, set the 'AxisDirection' value to 'Z'.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Z');
```

Plot the antenna's vertical polarization power pattern at 200 MHz as a 3-D polar plot.

```
fc = 200e6;
pattern(sSD,fc,[-180:180],[-90:90],...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Polarization','V')
```



As the above figure shows, the antenna pattern is that of a vertically-oriented dipole and has its maximum at the equator and nulls at the poles.

Short-Dipole Antenna Element Pattern over Selected Range

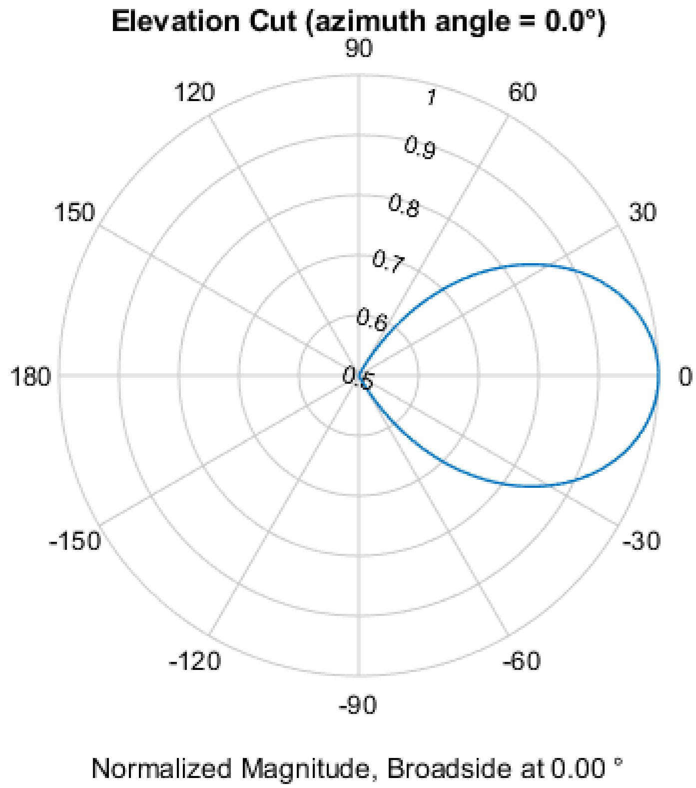
Specify a short-dipole antenna element with its dipole axis pointing along the z-axis. Then, plot the magnitude pattern over a selected range of angles. The antenna operating frequency spans the range 100 to 900 MHz.

To construct a z-directed short-dipole antenna, set the `'AxisDirection'` value to `'Z'`.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,...
    'AxisDirection','Z');
```

Plot the antenna's vertical polarization response at 200 MHz as an elevation cut at zero degrees azimuth angle. Restrict the plot from -60 to 60 degrees elevation in 0.1 degree increments.

```
fc = 200e6;
pattern(sSD,fc,0,[-60:0.1:60],...
    'CoordinateSystem','polar',...
    'Type','efield',...
    'Polarization','V')
```



Short-Dipole Antenna Element Directivity

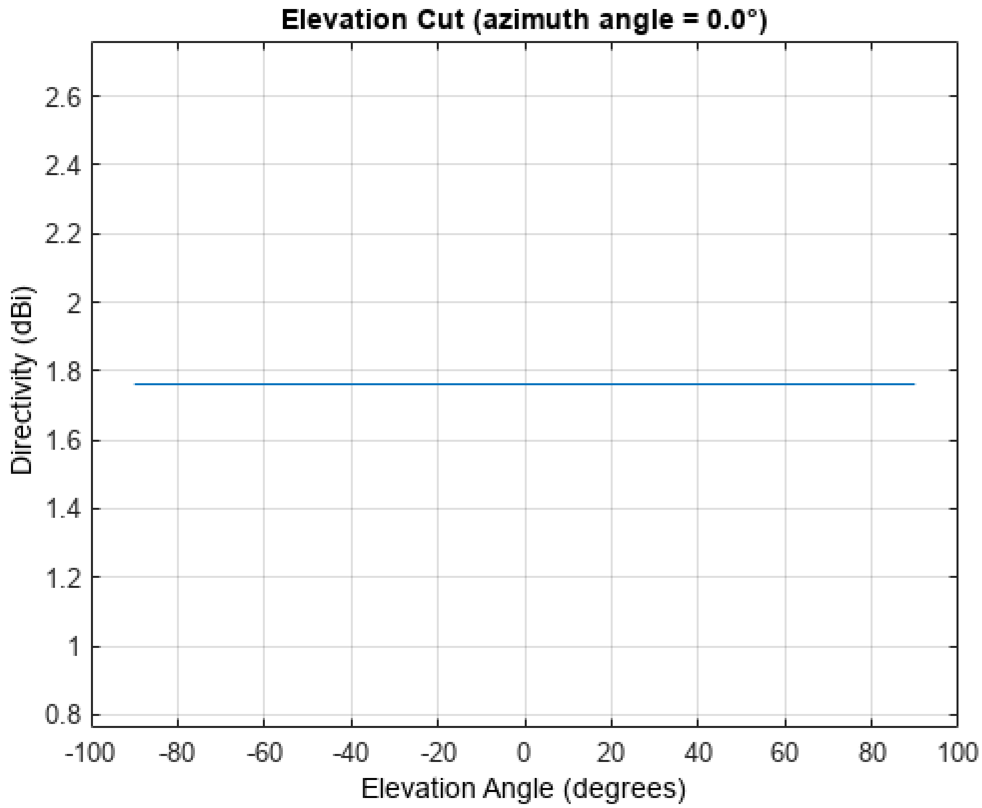
Specify a short-dipole antenna element with its dipole axis pointing along the y-axis. Then, plot the directivity. The antenna operating frequency spans the range 100 to 900 MHz.

Construct a y-directed short-dipole antenna by setting the `'AxisDirection'` value to `'Y'`.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,...
    'AxisDirection','Y');
```

Plot the antenna's directivity at 500 MHz as an elevation cut at zero degrees azimuth angle.

```
fc = 500e6;
pattern(sSD,fc,0,[-90:90],...
    'CoordinateSystem','rectangular',...
    'Type','directivity')
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.

Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.

Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.

'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the UV grid values using AZ and EL.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternElevation` | `patternAzimuth`

patternAzimuth

System object: phased.ShortDipoleAntennaElement
Package: phased

Plot short-dipole antenna element directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sElem,FREQ)
patternAzimuth(sElem,FREQ,EL)
patternAzimuth(sElem,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

patternAzimuth(sElem,FREQ) plots the 2-D element directivity pattern versus azimuth (in dBi) for the element sElem at zero degrees elevation angle. The argument FREQ specifies the operating frequency.

patternAzimuth(sElem,FREQ,EL), in addition, plots the 2-D element directivity pattern versus azimuth (in dBi) at the elevation angle specified by EL. When EL is a vector, multiple overlaid plots are created.

patternAzimuth(sElem,FREQ,EL,Name,Value) plots the element pattern with additional options specified by one or more Name,Value pair arguments.

PAT = patternAzimuth(___) returns the element pattern. PAT is a matrix whose entries represent the pattern at corresponding sampling points specified by the 'Azimuth' parameter and the EL input argument.

Input Arguments

sElem — Short-dipole antenna element
System object

Short-dipole antenna element, specified as a phased.ShortDipoleAntennaElement System object.

Example: sElem = phased.ShortDipoleAntennaElement;

FREQ — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

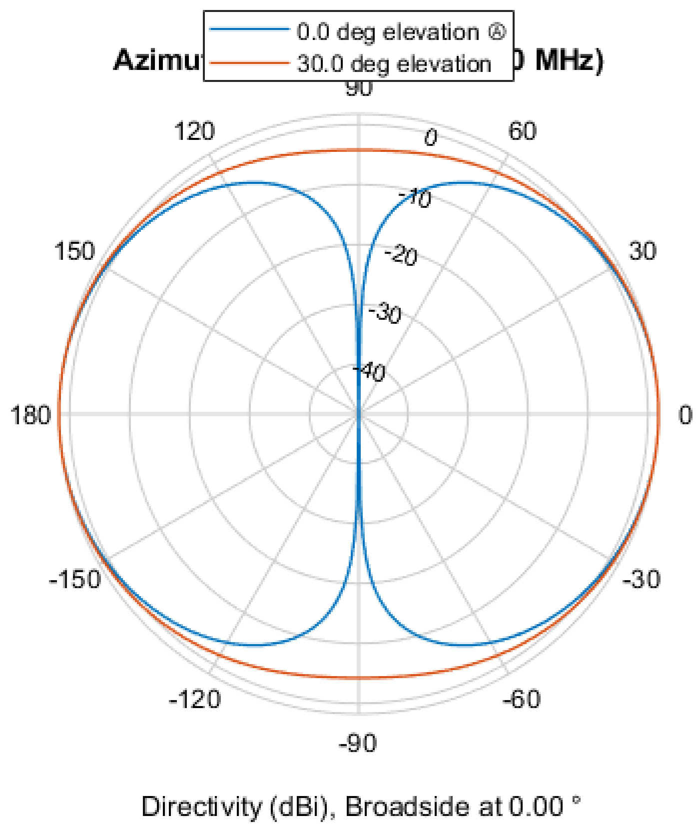
Examples

Azimuth Directivity of Short-Dipole Antenna Element at Two Elevations

Specify a short-dipole antenna element having a direction along the y-axis. Then, plot an azimuth cut of the directivity at 0 and 30 degrees elevation. Assume the operating frequency is 500 MHz.

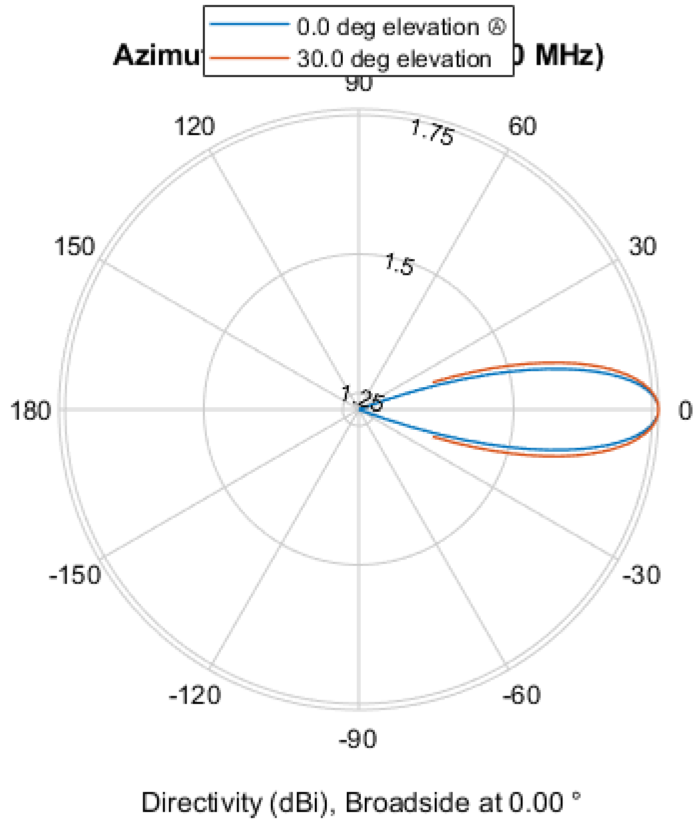
Create the antenna element.

```
fc = 500e6;
sSD = phased.ShortDipoleAntennaElement('FrequencyRange',[100,900]*1e6,...
    'AxisDirection','y');
patternAzimuth(sSD,fc,[0 30])
```



Plot a reduced range of azimuth angles using the `Azimuth` parameter. Notice the change in scale.

```
patternAzimuth(sSD,fc,[0 30],'Azimuth',[-20:20])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: phased.ShortDipoleAntennaElement
Package: phased

Plot short-dipole antenna element directivity or pattern versus elevation

Syntax

```
patternElevation(sElem,FREQ)
patternElevation(sElem,FREQ,AZ)
patternElevation(sElem,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

patternElevation(sElem,FREQ) plots the 2-D element directivity pattern versus elevation (in dBi) for the element sElem at zero degrees azimuth angle. The argument FREQ specifies the operating frequency.

patternElevation(sElem,FREQ,AZ), in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by AZ. When AZ is a vector, multiple overlaid plots are created.

patternElevation(sElem,FREQ,AZ,Name,Value) plots the element pattern with additional options specified by one or more Name,Value pair arguments.

PAT = patternElevation(___) returns the element pattern. PAT is a matrix whose entries represent the pattern at corresponding sampling points specified by the 'Elevation' parameter and the AZ input argument.

Input Arguments

sElem — Short-dipole antenna element
System object

Short-dipole antenna element, specified as a phased.ShortDipoleAntennaElement System object.

Example: sElem = phased.ShortDipoleAntennaElement;

FREQ — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

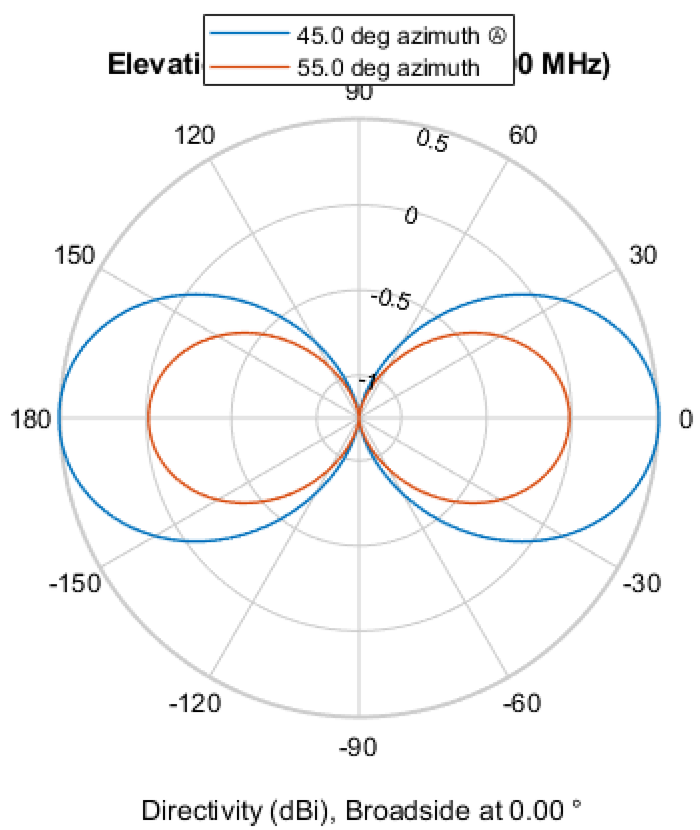
Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

Examples

Plot Elevation Pattern of Crossed-Dipole Antenna Element

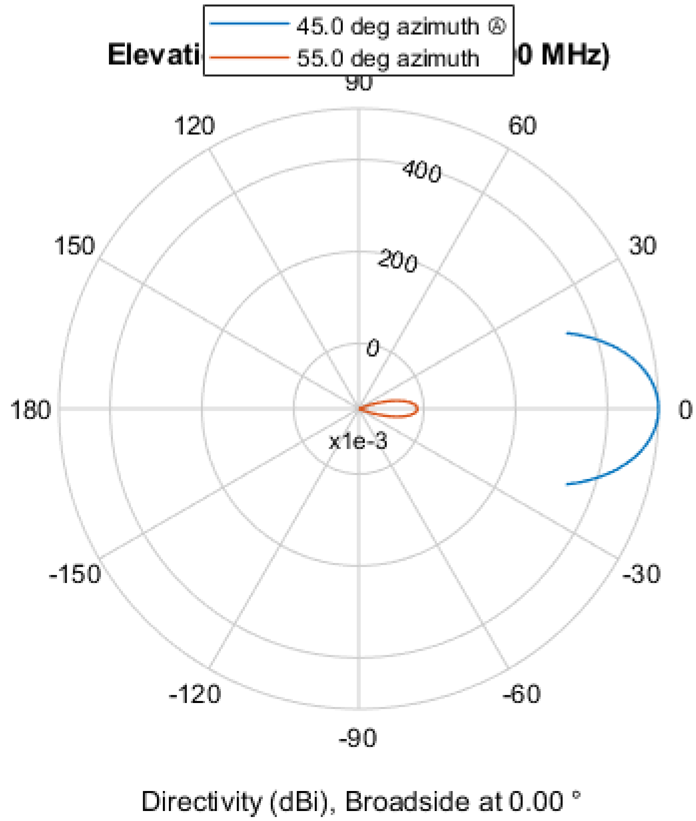
Plot the elevation directivity pattern of a crossed-dipole antenna at two different azimuths: 45° and 55°. Assume the operating frequency is 500 MHz.

```
fc = 500e6;
sCD = phased.CrossedDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
patternElevation(sCD,fc,[45 55])
```



Plot a reduced range of elevation angles using the `Elevation` parameter. Notice the change in scale.

```
patternElevation(sCD,fc,[45 55],'Elevation',-20:20)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also
`pattern` | `patternAzimuth`

plotResponse

System object: `phased.ShortDipoleAntennaElement`
Package: `phased`

Plot response pattern of antenna

Syntax

```
plotResponse(H,FREQ)
plotResponse(H,FREQ,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ)` plots the element response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`.

`plotResponse(H,FREQ,Name,Value)` plots the element response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

`H`

Element System object

`FREQ`

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CutAngle`

Cut angle specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

Format

Format of the plot, using one of 'Line', 'Polar', or 'UV'. If you set Format to 'UV', FREQ must be a scalar.

Default: 'Line'

NormalizeResponse

Set this value to true to normalize the response pattern. Set this value to false to plot the response pattern without normalizing it. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: true

OverlayFreq

Set this value to true to overlay pattern cuts in a 2-D line plot. Set this value to false to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is false, FREQ must be a vector with at least two entries.

This parameter applies only when Format is not 'Polar' and RespCut is not '3D'.

Default: true

Polarization

Specify the polarization options for plotting the antenna response pattern. The allowable values are |'None' | 'Combined' | 'H' | 'V' | where

- 'None' specifies plotting a nonpolarized response pattern
- 'Combined' specifies plotting a combined polarization response pattern
- 'H' specifies plotting the horizontal polarization response pattern
- 'V' specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is 'None'. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: 'None'

RespCut

Cut of the response. Valid values depend on Format, as follows:

- If Format is 'Line' or 'Polar', the valid values of RespCut are 'Az', 'El', and '3D'. The default is 'Az'.
- If Format is 'UV', the valid values of RespCut are 'U' and '3D'. The default is 'U'.

If you set RespCut to '3D', FREQ must be a scalar.

Unit

The unit of the plot. Valid values are 'db', 'mag', 'pow', or 'dbi'. This parameter determines the type of plot that is produced.

Unit value	Plot type
db	power pattern in dB scale
mag	field pattern
pow	power pattern
dbi	directivity

Default: 'db'

AzimuthAngles

Azimuth angles for plotting element response, specified as a row vector. The AzimuthAngles parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'Az' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the AzimuthAngles and ElevationAngles parameters simultaneously.

Default: [-180:180]

ElevationAngles

Elevation angles for plotting element response, specified as a row vector. The ElevationAngles parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the RespCut parameter is set to 'El' or '3D' and the Format parameter is set to 'Line' or 'Polar'. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the RespCut parameter to '3D', you can set the ElevationAngles and AzimuthAngles parameters simultaneously.

Default: [-90:90]

UGrid

U coordinate values for plotting element response, specified as a row vector. The UGrid parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to 'U' or '3D'. The values of UGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the UGrid and VGrid parameters simultaneously.

Default: [-1:0.01:1]

VGrid

V coordinate values for plotting element response, specified as a row vector. The VGrid parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to '3D'. The values of VGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the VGrid and UGrid parameters simultaneously.

Default: [-1:0.01:1]

Examples

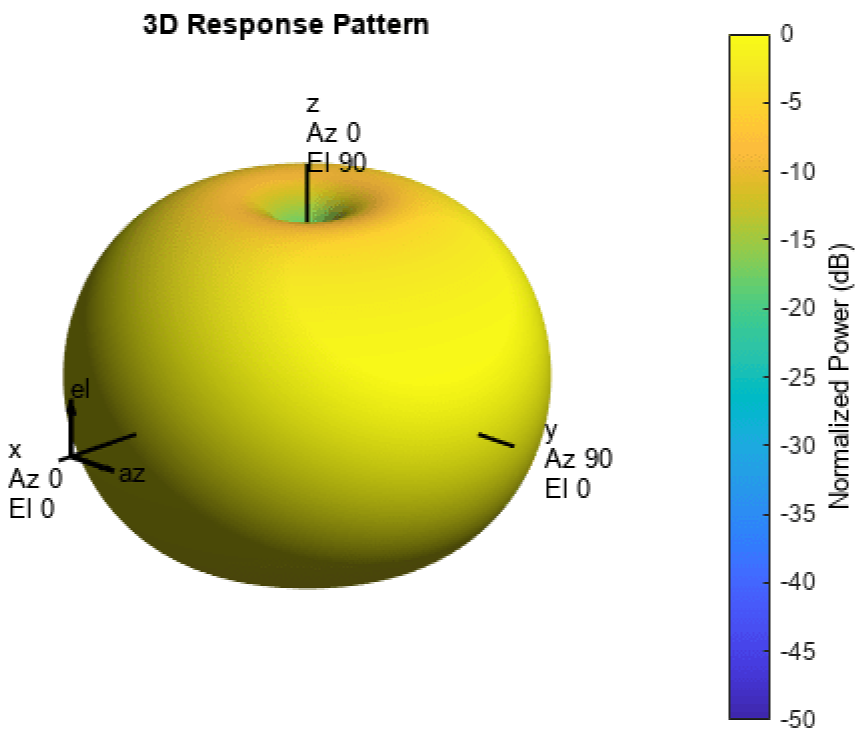
Response of Short-Dipole Antenna Oriented Along the Z-Axis

Specify a short-dipole antenna element with its dipole axis pointing along the z-axis. To do so, set the `'AxisDirection'` value to `'Z'`.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Z');
```

Plot the antenna's vertical polarization response at 200 MHz as a 3-D polar plot.

```
fc = 200e6;
plotResponse(sSD,fc,'Format','Polar',...
    'RespCut','3D','Polarization','V');
```



As the above figure shows, the antenna pattern is that of a vertically-oriented dipole and has its maximum at the equator and nulls at the poles.

Plot Short-Dipole Antenna Element Response over Selected Range

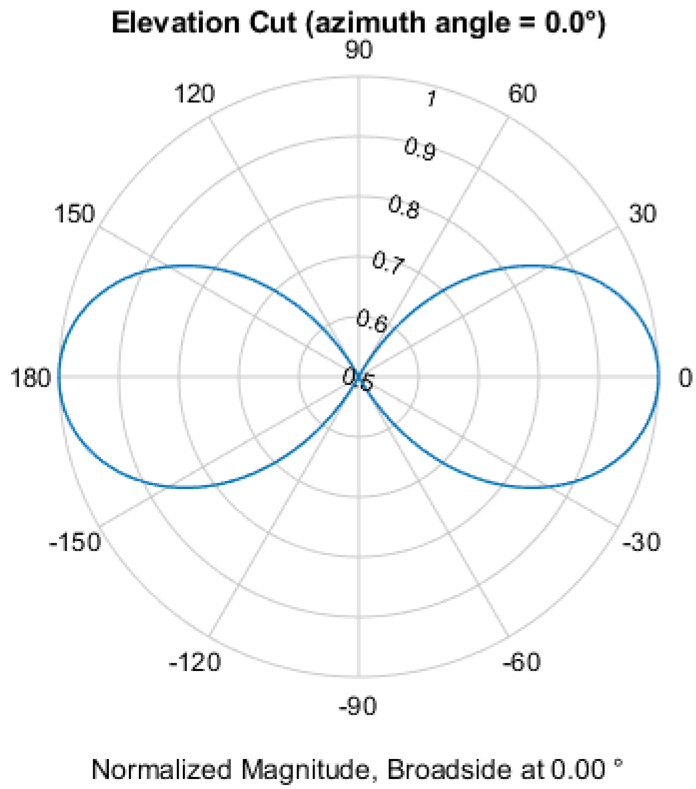
This example shows how to construct a short-dipole antenna element with its dipole axis pointing along the z-axis and how to plot the response over a selected range of angles. The antenna operating frequency spans the range 100 to 900 MHz.

To construct a z-directed short-dipole antenna, set the `'AxisDirection'` value to `'Z'`.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Z');
```

Plot the antenna's vertical polarization response at 200 MHz as an elevation cut at a fixed azimuth angle. Use the 'ElevationAngles' property to restrict the plot from -60 to 60 degrees elevation in 0.1 degree increments.

```
plotResponse(sSD,200e6,'Format','Polar',...
    'RespCut','El','Polarization','V',...
    'ElevationAngles',[-60:0.1:60],'Unit','mag');
```



Plot Short-Dipole Antenna Element Directivity

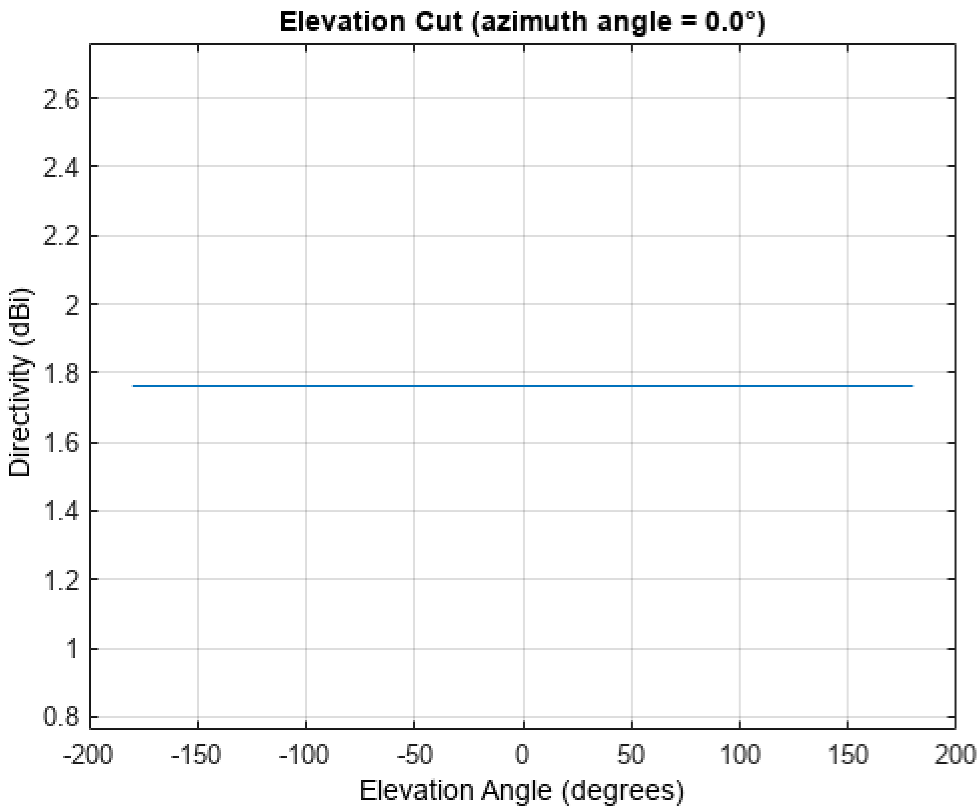
This example shows how to construct a short-dipole antenna element with its dipole axis pointing along the y-axis and how to plot the directivity. The antenna operating frequency spans the range 100 to 900 MHz.

To construct a y-directed short-dipole antenna, set the `'AxisDirection'` value to `'Y'`.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Y');
```

Plot the antenna's directivity at 500 MHz as an elevation cut at a fixed azimuth angle.

```
plotResponse(sSD,500e6,'Format','Line',...
    'RespCut','El','Unit','dbi');
```



See Also

uv2azel | azel2uv

step

System object: `phased.ShortDipoleAntennaElement`
Package: `phased`

Output response of antenna element

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the antenna's voltage response, `RESP`, at the operating frequencies specified in `FREQ` and in the directions specified in `ANG`. For the short-dipole antenna element object, `RESP` is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`, representing the horizontal and vertical polarization components of the antenna's response. Each field is an *M*-by-*L* matrix containing the antenna response at the *M* angles specified in `ANG` and at the *L* frequencies specified in `FREQ`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Antenna element object.

FREQ

Operating frequencies of antenna in hertz. `FREQ` is a row vector of length L.

ANG

Directions in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If ANG is a row vector of length M, each element specifies a direction's azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

Output Arguments

RESP

Voltage response of antenna element returned as a MATLAB struct with fields RESP.H and RESP.V. Both RESP.H and RESP.V contain responses for the horizontal and vertical polarization components of the antenna radiation pattern. Both RESP.H and RESP.V are M-by-L matrices. In these matrices, M represents the number of angles specified in ANG, and L represents the number of frequencies specified in FREQ.

Examples

Response of Short-Dipole Antenna

Find the response of a short-dipole antenna element at boresight, (0°,0°), and off boresight, (30°,0°). The antenna operates at 256 MHz.

```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Y');
ang = [0 30;0 0];
fc = 250e6;
resp = antenna(fc,ang)

resp = struct with fields:
    H: [2x1 double]
    V: [2x1 double]
```

Horizontal response.

```
disp(resp.H)

   -1.2247
   -1.0607
```

Vertical response.

```
disp(resp.V)

     0
     0
```

Algorithms

The total response of a short-dipole antenna element is a combination of its frequency response and spatial response. This System object calculates both responses using nearest neighbor interpolation and then multiplies the responses to form the total response.

See Also

`uv2azel` | `phitheta2azel`

phased.ScatteringMIMOChannel

Package: phased

Scattering MIMO channel

Description

The phased.ScatteringMIMOChannel System object models a multipath propagation channel in which radiated signals from a transmitting array are reflected from multiple scatterers back toward a receiving array. In this channel, propagation paths are line of sight from point to point. The object models range-dependent time delay, gain, Doppler shift, phase change, and atmospheric loss due to gases, rain, fog, and clouds.

The attenuation models for atmospheric gases and rain are valid for electromagnetic signals in the frequency range from 1 through 1000 GHz. The attenuation model for fog and clouds is valid from 10 through 1000 GHz. Outside of these frequency ranges, the object uses the nearest valid value.

To compute the multipath propagation for specified source and receiver points:

1. Define and set up your scattering MIMO channel using the “Construction” on page 1-1577 procedure. You can set the System object properties during construction or leave them at their default values.
2. Call the step method to compute the propagated signals using the properties of the phased.ScatteringMIMOChannel System object. You can change tunable properties before or after any call to the step method.

Note Instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

channel = phased.ScatteringMIMOChannel creates a scattering MIMO propagation channel System object, channel.

channel = phased.ScatteringMIMOChannel(Name,Value) creates a System object, channel, with each specified property Name set to the specified Value. You can specify additional name and value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

TransmitArray — Transmitting array

phased.ULA (default) | Phased Array System Toolbox antenna array System object

Transmitting array, specified as a Phased Array System Toolbox antenna array System object. The default value for this property is a phased.ULA array with its default property values.

Example: phased.URA

ReceiveArray — Receiving array
phased.ULA (default) | Phased Array System Toolbox antenna array System object

Receiving array, specified as a Phased Array System Toolbox antenna array System object. The default value for this property is a phased.ULA array with its default property values.

Example: phased.URA

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed'). See physconst for more information.

Example: 3e8

Data Types: double

CarrierFrequency — Signal carrier frequency
300e6 (default) | positive real-valued scalar

Signal carrier frequency, specified as a positive real-valued scalar. Units are in Hz.

Example: 100e6

Data Types: double

Polarization — Polarization configuration
'None' (default) | 'Combined' | 'Dual'

Polarization configuration, specified as 'None', 'Combined', or 'Dual'. When you set this property to 'None', the output field is considered a scalar field. When you set this property to 'Combined', the radiated fields are polarized and are interpreted as a single signal in the sensor's inherent polarization. When you set this property to 'Dual', the *H* and *V* polarization components of the radiated field are independent signals.

Example: 'Dual'

Data Types: char

SpecifyAtmosphere — Enable atmospheric attenuation model
false (default) | true

Option to enable the atmospheric attenuation model, specified as a false or true. Set this property to true to add signal attenuation caused by atmospheric gases, rain, fog, or clouds. Set this property to false to ignore atmospheric effects in propagation.

Setting SpecifyAtmosphere to true, enables the Temperature, DryAirPressure, WaterVapourDensity, LiquidWaterDensity, and RainRate properties.

Data Types: logical

Temperature — Ambient temperature
15 (default) | real-valued scalar

Ambient temperature, specified as a real-valued scalar. Units are in degrees Celsius.

Example: 20.0

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`DryAirPressure` — Atmospheric dry air pressure
`101.325e3` (default) | positive real-valued scalar

Atmospheric dry air pressure, specified as a positive real-valued scalar. Units are in pascals (Pa). The default value of this property corresponds to one standard atmosphere.

Example: `101.0e3`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`WaterVapourDensity` — Atmospheric water vapor density
`7.5` (default) | positive real-valued scalar

Atmospheric water vapor density, specified as a positive real-valued scalar. Units are in g/m^3.

Example: `7.4`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`LiquidWaterDensity` — Liquid water density
`0.0` (default) | nonnegative real-valued scalar

Liquid water density of fog or clouds, specified as a nonnegative real-valued scalar. Units are in g/m^3. Typical values for liquid water density are 0.05 for medium fog and 0.5 for thick fog.

Example: `0.1`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`RainRate` — Rainfall rate
`0.0` (default) | nonnegative scalar

Rainfall rate, specified as a nonnegative scalar. Units are in mm/hr.

Example: `10.0`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`SampleRate` — Sample rate of signal
`1e6` (default) | positive scalar

Sample rate of signal, specified as a positive scalar. Units are in Hz. The System object uses this quantity to calculate the propagation delay in units of samples.

Example: `1e6`

Data Types: `double`

`SimulateDirectPath` — Enable propagation along direct path
`false` (default) | `true`

Option to enable signal propagation along the direct path, specified as `false` or `true`. The direct path is a line-of-sight path from the transmitting array to the receiving array with no scattering.

Data Types: `logical`

`ChannelResponseOutputPort` — Enable output of channel response
`false` (default) | `true`

Option to enable output of channel response, specified as `false` or `true`. Set this property to `true`to output the channel response and time delay by using the `chmatrix` and `tau` output arguments of the `step` method.

Data Types: `logical`

`MaximumDelaySource` — Source of maximum delay
`'Auto'` (default) | `'Property'`

Source of the maximum delay value, specified as `'Auto'` or `'Property'`. When you set this property to `'Auto'`, the channel automatically allocates enough memory to simulate the propagation delay. When you set this property to `'Property'`, you can specify the maximum delay by using the `MaximumDelay` property. Signals arriving after the maximum delay are ignored.

`MaximumDelay` — Maximum signal delay
`10e-6` (default) | positive scalar

Maximum signal delay, specified as a positive scalar. Delays greater than this value are ignored. Units are in seconds.

Dependencies

To enable this property, set the `MaximumDelaySource` property to `'Property'`.

Data Types: `double`

`TransmitArrayMotionSource` — Source of transmitting array motion parameters
`'Property'` (default) | `'Input port'`

Source of the transmitting array motion parameters, specified as `'Property'` or `'Input port'`.

- When you set this property to `'Property'`, the transmitting array is stationary. Then, you can specify the location and orientation of the array using the `TransmitArrayPosition` and `TransmitArrayOrientationAxes` properties.
- When you set this property to `'Input port'`, specify the transmitting array location, velocity, and orientation by using the `txpos`, `txvel`, and `txaxes` input arguments of the `step` method.

Data Types: `char`

TransmitArrayPosition — Position of transmitting array phase center
[0;0;0] (default) | real-valued three-element vector

Position of the transmitting array phase center, specified as a real-valued three-element vector in Cartesian form, [x;y;z], with respect to the global coordinate system. Units are in meters.

Example: [1000;-200;55]

Dependencies

To enable this property, set the TransmitArrayMotionSource property to 'Property'.

Data Types: double

TransmitArrayOrientationAxes — Orientation of transmitting array
eye(3,3) (default) | real-valued 3-by-3 orthonormal matrix

Orientation of transmitting array, specified as a real-valued 3-by-3 orthonormal matrix. The matrix specifies the three axes, *(x,y,z)*, that define the local coordinate system of the array with respect to the global coordinate system. Matrix columns correspond to the axes of the local array coordinate system.

Example: rotz(45)

Dependencies

To enable this property, set the TransmitArrayMotionSource property to 'Property'.

Data Types: double

ReceiveArrayMotionSource — Source of receiving array motion parameters
'Property' (default) | 'Input port'

Source of the receiving array motion parameters, specified as 'Property' or 'Input port'.

- When you set this property to 'Property', the receiving array is stationary. Then, you can specify the location and orientation of the array by using the ReceiveArrayPosition and ReceiveArrayOrientationAxes properties.
- When you set this property to 'Input port', you can specify the receiving array location, velocity, and orientation by using the rxpos, rxvel, and rxaxes input arguments of the step method.

Data Types: char

ReceiveArrayPosition — Position of receiving array
[0;0;0] (default) | real-valued three-element vector

Position of the receiving array phase center, specified as a real-valued three-element vector in Cartesian form,[x;y;z], with respect to the global coordinate system. Units are in meters.

Example: [1000;-200;55]

Dependencies

To enable this property, set the ReceiveArrayMotionSource property to 'Property'.

Data Types: double

ReceiveArrayOrientationAxes — Orientation of receiving array
eye(3,3) (default) | real-valued 3-by-3 orthonormal matrix

Orientation of receiving array, specified as a real-valued 3-by-3 orthonormal matrix. The matrix specifies the three axes, *(x,y,z)*, that define the local coordinate system of the array with respect to the global coordinate system. Matrix columns correspond to the axes of the local array coordinate system.

Example: roty(60)

Dependencies

To enable this property, set the ReceiveArrayMotionSource property to 'Property'.

Data Types: double

ScattererSpecificationSource — Source of scatterer parameters
'Auto' (default) | 'Property' | 'Input port'

Source of scatterer parameters, specified as 'Auto', 'Property', 'Input port'.

- When you set this property to 'Auto', all scatterer positions and coefficients are randomly generated. Scatterer velocities are zero. The generated positions are contained within the region defined by the ScattererPositionBoundary. To set the number of scatterers, use the NumScatterers property.
- When you set this property to 'Property', you can set the scatterer positions by using the ScattererPosition property and the scattering coefficients by using the ScattererCoefficient property. All scatterer velocities are zero.
- When you set this property to 'Input port', you can specify the scatterer positions, velocities, and scattering coefficients using the scatpos, scatvel, and scatcoef input arguments of the step method.

Example: 'Input port'

Data Types: char

NumScatterers — Number of scatterers
1 (default) | nonnegative integer

Number of scatterers, specified as a nonnegative integer.

Example: 9

Dependencies

To enable this property, set the ScattererSpecificationSource property to 'Auto'.

Data Types: double

ScattererPositionBoundary — Boundary of scatterer positions
[0,1000] (default) | 1-by-2 real-valued vector | 3-by-2 real-valued matrix

Boundary of the scatterer positions, specified as a 1-by-2 real-valued row vector or a 3-by-2 real-valued matrix. The vector specifies the minimum and maximum, [minbdry maxbdry], for all three dimensions. The matrix specifies boundaries in all three dimensions in the form [x_minbdry x_maxbdry;y_minbdry y_maxbdry; z_minbdry z_maxbdry].

Example: [-1000 500;-100 100;-200 0]

Dependencies

To enable this property, set the `ScattererSpecificationSource` property to `'Auto'`.

Data Types: `double`

`ScattererPosition` — Positions of scatterers
`[0;0;0]` (default) | real-valued 3-by-K matrix

Positions of the scatterers, specified as real-valued 3-by-K matrix. K is the number of scatterers. Each column represents a different scatterer and has the Cartesian form `[x;y;z]` with respect to the global coordinate system. Units are in meters.

Example: `[1050 -100;-300 55;0 -75]`

Dependencies

To enable this property, set the `ScattererSpecificationSource` property to `'Property'`.

Data Types: `double`

`ScattererCoefficient` — Scattering coefficients
`1` (default) | complex-valued 1-by-K vector

Scattering coefficients, specified as a complex-valued 1-by-K vector. K is the number of scatterers. Units are dimensionless.

Example: `2+1i`

Dependencies

To enable this property, set the `ScattererSpecificationSource` property to `'Property'`.

Data Types: `double`

`ScatteringMatrix` — Scattering matrices
`[1 0;0 1]` | complex–valued 2-by-2-by-N_s array

Scattering matrices of the scatterers, specified as a complex–valued 2-by-2-by-N_s array where N_s is the number of scatterers. Each page of this array represents the scattering matrix of a scatterer. Each scattering matrix has the form `[s_hh s_hv;s_vh s_vv]`. For example, the component `s_hv` specifies the complex scattering response when the input signal is vertically polarized and the reflected signal is horizontally polarized. The other components are defined similarly. Units are in square meters.

Dependencies

To enable this property, set the `ScatteringMatrixSource` property to `'Property'` and the `Polarization` property to `'Combined'` or `'Dual'`.

Data Types: `double`

`ScattererOrientationAxes` — Orientation of scatterers
`[1 0 0;0 1 0;0 0 1]` (default) | real–valued 3-by-3-by-N_s array

Orientation of the scatterers, specified as a real–valued 3-by-3-by-N_s array where N_s is the number of scatterers. Each page of this array is an orthonormal matrix. Matrix columns represent the axis of the local coordinates (x,y,z) of the scatterer with respect to the global coordinate system.

Example: `roty(45)`

Dependencies

To enable this property, set the ScatteringMatrixSource property to 'Property' and the Polarization property to 'Combined' or 'Dual'.

Data Types: double

SeedSource — Source of random number generator seed
'Auto' (default) | 'Property'

Source of random number generator seed, specified as 'Auto' or 'Property'.

- When you set this property to 'Auto', random numbers are generated using the default MATLAB random number generator.
- When you set this property to 'Property', the object uses a private random number generator with the seed specified by the value of the Seed property.

To use this object with Parallel Computing Toolbox software, set this property to 'Auto'.

Dependencies

To enable this property, set the ScattererSpecificationSource property to 'Auto'.

Seed — Random number generator seed
0 (default) | nonnegative integer

Random number generator seed, specified as a nonnegative integer less than 2^{32}.

Example: 5005

Dependencies

To enable this property, set the ScattererSpecificationSource property to 'Auto' and the SeedSource property to 'Property'.

Data Types: double

Methods

reset	Reset state of the System object
step	Propagate signals in scattering MIMO channel

Common to All System Objects	
release	Allow System object property value changes

Examples

Propagate Signals in MIMO Channel

Create a 30 GHz MIMO channel with random scatterers. The scenario contains a stationary 21-element transmitting ULA array and a stationary 15-element receiving ULA array. The transmitting antennas have cosine responses and the receiving antennas are isotropic. Element spacing for both arrays is less than one-half wavelength. The channel has 50 randomly generated static scatterers within a specified bounding box. The transmit array is located at [0;20;50] meters and the receive array is located at [200;10;10] meters. Compute the propagated signal through this channel. The sample rate for the signal is 10 MHz.

```
fc = 30e9;
c = physconst('LightSpeed');
lambda = c/fc;
fs = 10e6;
txarray = phased.ULA('Element',phased.CosineAntennaElement,...
    'NumElements',21,'ElementSpacing',0.45*lambda);
rxarray = phased.ULA('Element',phased.IsotropicAntennaElement,...
    'NumElements',15,'ElementSpacing',0.45*lambda);

channel = phased.ScatteringMIMOChannel('TransmitArray',txarray,...
    'ReceiveArray',rxarray,'PropagationSpeed',c,'CarrierFrequency',fc,...
    'SampleRate',fs,'TransmitArrayPosition',[0;20;50],...
    'ReceiveArrayPosition',[200;10;10],'NumScatterers',50,...
    'ScattererPositionBoundary',[10 180; -30 30; -30 30]);
```

Create a random data signal of ones and zeros for each transmitter.

```
x = randi(2,[100 21]) - 1;
```

Compute the received signals after propagating through the channel.

```
y = channel(x);
```

Propagate Signals in MIMO Channel from Moving Transmitter

Create a MIMO channel containing 3 fixed scatterers. The scenario contains a 21-element transmitting ULA array operating at 72 GHz, and a 15-element receiving ULA array. The transmitting elements have cosine response shapes and the receiving antennas are isotropic. Only the transmitting antenna is moving. Element spacing for both arrays is less than one-half wavelength. The transmitting array starts at (0,20,50) meters and moves towards the receiver at 2 m/s. The receiving array is located at (200,10,10) meters. Compute the propagated signal through this channel. The sample rate for the signal is 10 MHz.

```
fc = 72e9;
c = physconst('LightSpeed');
lambda = c/fc;
fs = 10e6;
txplatform = phased.Platform('MotionModel','Velocity','InitialPosition', ...
    [0;20;50],'Velocity',[2;0;0]);
txarray = phased.ULA('Element',phased.CosineAntennaElement, ...
    'NumElements',21,'ElementSpacing',0.45*lambda);
rxarray = phased.ULA('Element',phased.IsotropicAntennaElement, ...
    'NumElements',15,'ElementSpacing',0.45*lambda);
channel = phased.ScatteringMIMOChannel('TransmitArray',txarray, ...
    'ReceiveArray',rxarray,'PropagationSpeed',c,'CarrierFrequency',fc,...
    'SampleRate',fs,'TransmitArrayMotionSource','Input port', ...
    'ReceiveArrayMotionSource','Property','ReceiveArrayPosition',[200;10;10],...
    'ReceiveArrayOrientationAxes',rotz(180),...
    'ScattererSpecificationSource','Property','ScattererPosition', ...
    [75 100 120; -10 20 12; 5 -5 8],'ScattererCoefficient',[1i,2+3i,-1+1i]);
```

Move the platforms for two time steps at one second intervals. For each time instance:

- Create a random data signal of ones and zeros for each transmitter element.
- Move the transmitter and receiver. The orientations are fixed.
- Propagate the signals from transmitters to scatterers to receiver.

```
for k =1:2
    x = randi(2,[100 21]) - 1;
    [txpos,txvel] = txplatform(1);
    txaxes = eye(3);
    y = channel(x,txpos,txvel,txaxes);
end
```

Propagate Signals Through MIMO Channel to Moving Receiver

Create a MIMO channel containing 3 fixed scatterers. The scenario contains a 21-element transmitting ULA array and a 15-element receiving ULA array. Both arrays operate at 72 GHz. The transmitting elements have cosine response shapes and the receiving antennas are isotropic. Only the receiving antenna is moving. Element spacing for both arrays is less than one-half wavelength. The transmitting array is located at (0,20,50) meters. The receiving array starts at (200,10,10) meters and moves toward the transmitter at 2 m/s. Compute the propagated signal through this channel. The sample rate for the signal is 10 MHz.

```
fc = 72e9;
c = physconst('LightSpeed');
lambda = c/fc;
fs = 10e6;
rxplatform = phased.Platform('MotionModel','Velocity','InitialPosition',...
    [200;10;10],'Velocity',[-2;0;0]);
txarray = phased.ULA('Element',phased.CosineAntennaElement, ...
    'NumElements',21,'ElementSpacing',0.45*lambda);
rxarray = phased.ULA('Element',phased.IsotropicAntennaElement, ...
    'NumElements',15,'ElementSpacing',0.45*lambda);
channel = phased.ScatteringMIMOChannel('TransmitArray',txarray, ...
    'ReceiveArray',rxarray,'PropagationSpeed',c,'CarrierFrequency',fc, ...
    'SampleRate',fs,'TransmitArrayMotionSource','Property',...
    'TransmitArrayPosition',[0;20;50],'TransmitArrayOrientationAxes',eye(3,3), ...
    'ReceiveArrayMotionSource','Input port','ScattererSpecificationSource', ...
    'Property','ScattererPosition',[75 100 120; -10 20 12; 5 -5 8], ...
    'ScattererCoefficient',[1i,2+3i,-1+1i],'SpecifyAtmosphere',false);
```

Move the platforms for two time steps at one-second intervals. For each time instance:

- Create a random data signal of ones and zeros for each transmitter element.
- Move the transmitter and receiver. Fix the array orientations.
- Propagate the signals from transmitters to scatterers to receiver.

```
for k =1:2
    x = randi(2,[100 21]) - 1;
    [rxpos,rxvel] = rxplatform(1);
    rxaxes = rotz(45);
    y = channel(x,rxpos,rxvel,rxaxes);
end
```

Propagate Polarized Signals in MIMO Channel

Create a MIMO channel at 30 GHz with a 16-element transmit array and a 64-element receive array. Assume the elements are short-dipole antennas and the arrays are uniform linear arrays. The transmit array is located at [0;0;50] meters.

The receive array has an initial position at [200;0;0] meters and is moving at a speed of [10;0;0] meters/second. There are 200 static scatterers randomly located on the *xy* plane within a square centered at [200;0;0] and with a side length of 100 meters.

Use the channel to compute the propagated polarized signal. Assume the sample rate for the signal is 10 MHz and the frame length is 1000 samples. Collect 5 frames of received signal.

```
fc = 30e9;
c = 3e8;
lambda = c/fc;
fs = 10e6;
txarray = phased.ULA('Element',phased.ShortDipoleAntennaElement,...
    'NumElements',16,'ElementSpacing',lambda/2);
rxarray = phased.ULA('Element',phased.ShortDipoleAntennaElement,...
    'NumElements',64,'ElementSpacing',lambda/2);

Ns = 200;
scatpos = [100*rand(1,Ns) + 150; 100*rand(1,Ns) + 150; zeros(1,Ns)];
temp = randn(1,Ns) + 1i*randn(1,Ns);
scatcoef = repmat(eye(2),1,1,Ns).*permute(temp,[1 3 2]);
scatax = repmat(eye(3),1,1,Ns);

Nframesamp = 1000;
Tframe = Nframesamp/fs;
rxmobile = phased.Platform('InitialPosition',[200;0;0],...
    'Velocity',[10;0;0],'OrientationAxesOutputPort',true);

chan = phased.ScatteringMIMOChannel(...
    'TransmitArray',txarray,...
    'ReceiveArray',rxarray,...
    'PropagationSpeed',c,...
    'CarrierFrequency',fc,...
    'SampleRate',fs,...
    'Polarization','Dual',...
    'TransmitArrayPosition',[0;0;50],...
    'ReceiveArrayMotionSource','Input port',...
    'ScattererSpecificationSource','Property',...
    'ScattererPosition',scatpos,...
    'ScatteringMatrix',scatcoef,...
    'ScattererOrientationAxes',scatax);

xh = randi(2,[Nframesamp 16])-1;
xv = randi(2,[Nframesamp 16])-1;

for m = 1:5
    [rxpos,rxvel,rxax] = rxmobile(Tframe);
    [yh,yv] = chan(xh,xv,rxpos,rxvel,rxax);
end
```

More About

Attenuation and Loss Factors

Attenuation or path loss in the scattering MIMO channel consists of four components. $L = L_{fsp}L_gL_cL_r$, where:

- L_{fsp} is the free-space path attenuation.
- L_g is the atmospheric path attenuation.
- L_c is the fog and cloud path attenuation.
- L_r is the rain path attenuation.

Each component is in magnitude units, not in dB.

Free-Space Time Delay and Loss

When the origin and destination are stationary relative to each other, you can write the output signal of a free-space channel as $Y(t) = x(t\text{-}\tau)/L_{fsp}$. The quantity τ is the signal delay and L_{fsp} is the free-space path loss. The delay τ is given by R/c, where R is the propagation distance and c is the propagation speed. The free-space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far field of the transmitting element or array. In the near field, the free-space path loss formula is not valid and can result in a loss smaller than one, equivalent to a signal gain. Therefore, the loss is set to unity for range values, $R \le \lambda/4\pi$.

When the origin and destination have relative motion, the processing also introduces a Doppler frequency shift. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The quantity v is the relative speed of the destination with respect to the origin.

For more details on free space channel propagation, see [8]

Atmospheric Gas Attenuation Model

This model calculates the attenuation of signals that propagate through atmospheric gases.

Electromagnetic signals attenuate when they propagate through the atmosphere. This effect is due primarily to the absorption resonance lines of oxygen and water vapor, with smaller contributions coming from nitrogen gas. The model also includes a continuous absorption spectrum below 10 GHz. The ITU model *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* is used. The model computes the specific attenuation (attenuation per kilometer) as a function of temperature, pressure, water vapor density, and signal frequency. The atmospheric gas model is valid for frequencies from 1–1000 GHz and applies to polarized and nonpolarized fields.

The formula for specific attenuation at each frequency is

$$\gamma = \gamma_o(f) + \gamma_w(f) = 0.1820 f N''(f) .$$

The quantity $N''()$ is the imaginary part of the complex atmospheric refractivity and consists of a spectral line component and a continuous component:

$$N''(f) = \sum_i S_i F_i + N''_D(f)$$

The spectral component consists of a sum of discrete spectrum terms composed of a localized frequency bandwidth function, $F(f)_i$, multiplied by a spectral line strength, S_i. For atmospheric oxygen, each spectral line strength is

$$S_i = a_1 \times 10^{-7}\left(\frac{300}{T}\right)^3 \exp\left[a_2(1 - \left(\frac{300}{T}\right)\right]P\,.$$

For atmospheric water vapor, each spectral line strength is

$$S_i = b_1 \times 10^{-1}\left(\frac{300}{T}\right)^{3.5} \exp\left[b_2(1 - \left(\frac{300}{T}\right)\right]W\,.$$

P is the dry air pressure, W is the water vapor partial pressure, and T is the ambient temperature. Pressure units are in hectoPascals (hPa) and temperature is in degrees Kelvin. The water vapor partial pressure, W, is related to the water vapor density, ρ, by

$$W = \frac{\rho T}{216.7}\,.$$

The total atmospheric pressure is $P + W$.

For each oxygen line, S_i depends on two parameters, a_1 and a_2. Similarly, each water vapor line depends on two parameters, b_1 and b_2. The ITU documentation cited at the end of this section contains tabulations of these parameters as functions of frequency.

The localized frequency bandwidth functions $F_i(f)$ are complicated functions of frequency described in the ITU references cited below. The functions depend on empirical model parameters that are also tabulated in the reference.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length, R. Then, the total attenuation is $L_g = R(\gamma_o + \gamma_w)$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

For a complete description of this model, see [4].

Fog and Cloud Attenuation Model

This model calculates the attenuation of signals that propagate through fog or clouds.

Fog and cloud attenuation are the same atmospheric phenomenon. The ITU model, *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog* is used. The model computes the specific attenuation (attenuation per kilometer), of a signal as a function of liquid water density, signal frequency, and temperature. The model applies to polarized and nonpolarized fields. The formula for specific attenuation at each frequency is

$$\gamma_c = K_l(f)M,$$

where M is the liquid water density in gm/m^3. The quantity $K_l(f)$ is the specific attenuation coefficient and depends on frequency. The cloud and fog attenuation model is valid for frequencies 10–1000 GHz. Units for the specific attenuation coefficient are (dB/km)/(g/m^3).

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length R. Total attenuation is $L_c = R\gamma_c$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply narrowband attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

For a complete description of this model, see [5]

Rainfall Attenuation Model

This model calculates the attenuation of signals that propagate through regions of rainfall. Rain attenuation is a dominant fading mechanism and can vary from location-to-location and from year-to-year.

Electromagnetic signals are attenuated when propagating through a region of rainfall. Rainfall attenuation is computed according to the ITU rainfall model *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. The model computes the specific attenuation (attenuation per kilometer) of a signal as a function of rainfall rate, signal frequency, polarization, and path elevation angle. The specific attenuation, γ_{R}, is modeled as a power law with respect to rain rate

$$\gamma_R = kR^{\alpha},$$

where R is rain rate. Units are in mm/hr. The parameter k and exponent α depend on the frequency, the polarization state, and the elevation angle of the signal path. The specific attenuation model is valid for frequencies from 1–1000 GHz.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the an effective propagation distance, d_{eff}. Then, the total attenuation is $L = d_{\mathrm{eff}}\gamma_{\mathrm{R}}$.

The effective distance is the geometric distance, d, multiplied by a scale factor

$$r = \frac{1}{0.477d^{0.633}R_{0.01}^{0.073\alpha}f^{0.123} - 10.579(1 - \exp(-0.024d))}$$

where f is the frequency. The article *Recommendation ITU-R P.530-17 (12/2017): Propagation data and prediction methods required for the design of terrestrial line-of-sight systems* presents a complete discussion for computing attenuation.

The rain rate, R, used in these computations is the long-term statistical rain rate, $R_{0.01}$. This is the rain rate that is exceeded 0.01% of the time. The calculation of the statistical rain rate is discussed in *Recommendation ITU-R P.837-7 (06/2017): Characteristics of precipitation for propagation modelling*. This article also explains how to compute the attenuation for other percentages from the 0.01% value.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Version History

Introduced in R2017a

References

[1] Heath, R. Jr. et al. “An Overview of Signal Processing Techniques for Millimeter Wave MIMO Systems”, arXiv.org:1512.03007 [cs.IT], 2015.

[2] Tse, D. and P. Viswanath, *Fundamentals of Wireless Communications*, Cambridge: Cambridge University Press, 2005.

[3] Paulraj, A. *Introduction to Space-Time Wireless Communications*, Cambridge: Cambridge University Press, 2003.

[4] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases*. 2013.

[5] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog*. 2013.

[6] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. 2005.

[7] Seybold, J. *Introduction to RF Propagation*. New York: Wiley & Sons, 2005.

[8] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Functions
`rangeangle` | `fogpl` | `gaspl` | `rainpl` | `fspl` | `diagbfweights` | `scatteringchanmtx` | `waterfill`

Objects
`phased.FreeSpace` | `phased.RadarTarget` | `phased.BackscatterRadarTarget` | `twoRayChannel` | `phased.LOSChannel`

reset

System object: `phased.ScatteringMIMOChannel`
Package: `phased`

Reset state of the System object

Syntax

`reset(channel)`

Description

`reset(channel)` resets the internal state of the `phased.ScatteringMIMOChannel` System object, `channel`.

Input Arguments

`channel` — Spatial MIMO channel
`phased.ScatteringMIMOChannel` System object

Scattering MIMO channel, specified as a `phased.ScatteringMIMOChannel` System object.

Version History

Introduced in R2017a

step

System object: `phased.ScatteringMIMOChannel`
Package: `phased`

Propagate signals in scattering MIMO channel

Syntax

```
Y = step(channel,X)
[YH,YV] = step(channel,[XH,XV])
[ ___ ] = step( ___ ,txpos,txvel,txaxes)
[ ___ ] = step( ___ ,rxpos,rxvel,rxaxes)
[ ___ ] = step( ___ ,scatpos,scatvel,scatcoef)
[ ___ ] = step( ___ ,scatpos,scatvel,scatmat,scataxes)
[ ___ ,CR,TAU] = step(channel, ___ )
[ ___ ,CR_HH,CR_HV,CR_VH,CR_V,TAU] = step(channel, ___ )
[Y,CR,TAU] = step(channel,X,txpos,txvel,txaxesrxpos,rxvel,rxaxes,scatpos,
scatvel,scatcoef)
```

Description

Note Instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `Y = step(obj,X)` and `Y = obj(X)` perform equivalent operations.

`Y = step(channel,X)` uses the scattering MIMO channel, `channel`, to propagate a signal, `X`, from a transmitting array towards multiple scatterers, and returns the scattered signals, `Y`, to a receiving array.

To enable this syntax, set the `TransmitArrayMotionSource`, `ReceiveArrayMotionSource`, and `ScattererSpecificationSource` properties to `'Property'`.

`[YH,YV] = step(channel,[XH,XV])` propagates the polarized signals, `XH` and `XV`, through the *H*-port and *V*-port of the transmit array. The object returns the received signals, `YH` and `YV` to the *H*-port and *V*-port of the receive array.

To enable this syntax, set the `Polarization` property to `'Dual'`.

`[ ___ ] = step( ___ ,txpos,txvel,txaxes)` also specifies the transmitting array position, velocity, and axes orientation.

To enable this syntax, set the `ReceiveArrayMotionSource` and `ScattererSpecificationSource` properties to `'Property'` and set `TransmitArrayMotionSource` to `'Input port'`.

`[ ___ ] = step( ___ ,rxpos,rxvel,rxaxes)` specifies the receiving array position, velocity, and axes orientation.

To enable this syntax, set the TransmitArrayMotionSource and ScattererSpecificationSource properties to 'Property' and set ReceiveArrayMotionSource to 'Input port'.

[___] = step(___ ,scatpos,scatvel,scatcoef) specifies the scatterer positions and velocities, and the scattering coefficients.

To enable this syntax, set the TransmitArrayMotionSource and ReceiveArrayMotionSource properties to 'Property', set ScattererSpecificationSource to 'Input port', and set the Polarization property to 'None'.

[___] = step(___ ,scatpos,scatvel,scatmat,scataxes) specifies the scatterer positions, scatpos, and velocities, scatvel, the scattering matrix, scatmat, and the scatterer orientation axes, scataxes.

To enable this syntax, set the TransmitArrayMotionSource and ReceiveArrayMotionSource properties to 'Property', set ScattererSpecificationSource to 'Input port', and set the Polarization property to 'Combined' or 'Dual'.

[___ ,CR,TAU] = step(channel, ___) also returns the channel response matrix, CR, and the channel path delays, TAU, using any of the previous input argument combinations.

To enable this syntax, set the ChannelResponseOutputPort property to true and set the Polarization property to 'None' or 'Combined'.

[___ ,CR_HH,CR_HV,CR_VH,CR_V,TAU] = step(channel, ___) also returns the channel response matrices, CR_HH, CR_HV, CR_VH, and CR_V, using any of the previous input argument combinations.

To enable this syntax, set the ChannelResponseOutputPort property to true and set the Polarization property to 'Dual'.

You can combine optional input arguments when their enabling properties are set. For example, [Y, CR,TAU] = step(channel,X,txpos,txvel,txaxesrxpos,rxvel,rxaxes,scatpos, scatvel,scatcoef).

Note The object performs an initialization the first time the step method is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Input Arguments

channel — Scattering MIMO channel

phased.ScatteringMIMOChannel System object

Scattering MIMO channel, specified as a phased.ScatteringMIMOChannel System object.

Example: phased.ScatteringMIMOChannel

X — Transmitted narrowband signal

M-by-N_t complex-valued matrix

Transmitted narrowband signal, specified as an M-by-N_t complex-valued matrix. The quantity M is the number of samples in the signal, and N_t is the number of transmitting array elements. Each column represents the signal transmitted by the corresponding array element.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Dependencies

To enable this argument, set the `Polarization` property to `'None'` or `'Combined'`.

Data Types: `double`
Complex Number Support: Yes

`XH` — Transmitted narrowband H-polarization signal

M-by-N_t complex-valued matrix

Transmitted narrowband H-polarization signal, specified as an M-by-N_t complex-valued matrix. The quantity M is the number of samples in the signal, and N_t is the number of transmitting array elements. Each column represents the signal transmitted by the corresponding array element.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Dependencies

To enable this argument, set the `Polarization` property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

`XV` — Transmitted narrowband V-polarization signal

M-by-N_t complex-valued matrix

Transmitted narrowband V-polarization signal, specified as an M-by-N_t complex-valued matrix. The quantity M is the number of samples in the signal, and N_t is the number of transmitting array elements. Each column represents the signal transmitted by the corresponding array element.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Dependencies

To enable this argument, set the `Polarization` property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

`txpos` — Position of transmitting antenna array

real-valued three-element column vector

Position of transmitting antenna array, specified as real-valued three-element column vector taking the form `[x;y;z]`. The vector elements correspond to the *x*, *y*, and *z* positions of the array. Units are in meters.

Example: `[1000;100;500]`

Dependencies

To enable this argument, set the `TransmitArrayMotionSource` property to `'Input port'`.

Data Types: `double`

`txvel` — Velocity of transmitting antenna array
real-valued three-element column vector

Velocity of transmitting antenna array, specified as a real-valued three-element column vector taking the form `[vx;vy;vz]`. The vector elements correspond to the *x*, *y*, and *z* velocities of the array. Units are in meters per second.

Example: `[10;0;5]`

Dependencies

To enable this argument, set the `TransmitArrayMotionSource` property to `'Input port'`.

Data Types: `double`

`txaxes` — Axes orientation of transmitting antenna array
real-valued 3-by-3 real orthonormal matrix

Axes orientation of transmitting antenna array, specified as a real-valued 3-by-3 real orthonormal matrix. The matrix defines the orientation of the array local coordinate system with respect to the global coordinates. Matrix columns correspond to the directions of the *x*, *y*, and *z* axes of the local coordinate system. Units are dimensionless.

Example: `rotx(35)`

Dependencies

To enable this argument, set the `TransmitArrayMotionSource` property to `'Input port'`.

Data Types: `double`

`rxpos` — Position of receiving antenna array
real-valued three-element column vector

Position of receiving antenna array, specified as a real-valued three-element column vector taking the form `[x;y;z]`. The vector elements correspond to the *x*, *y*, and *z* positions of the array. Units are in meters.

Example: `[1000;100;500]`

Dependencies

To enable this argument, set the `ReceiveArrayMotionSource` property to `'Input port'`.

Data Types: `double`

`rxvel` — Velocity of receiving antenna array
real-valued three-element column vector

Velocity of receiving antenna array, specified as a real-valued three-element column vector taking the form `[vx;vy;vz]`. The vector elements correspond to the *x*, *y*, and *z* velocities of the array. Units are in meters per second.

Example: `[10;0;5]`

Dependencies

To enable this argument, set the `ReceiveArrayMotionSource` property to `'Input port'`.

Data Types: `double`

`rxaxes` — Axes orientation of receiving antenna array
real-valued 3-by-3 real orthonormal matrix

Axes orientation of receiving antenna array, specified as a real-valued 3-by-3 real orthonormal matrix. The matrix defines the orientation of the array local coordinate system with respect to the global coordinates. Matrix columns correspond to the directions of the *x*, *y*, and *z* axes of the local coordinate system. Units are dimensionless.

Example: `rotx(35)`

Dependencies

To enable this argument, set the `ReceiveArrayMotionSource` property to `'Input port'`.

Data Types: `double`

`scatpos` — Positions of scatterers
real-valued 3-by-N_s matrix

Positions of scatterers, specified as a real-valued 3-by-N_s matrix. The matrix contains the *(x,y,z)* positions of scatterers. Each column of the matrix specifies a different scatterer and takes the form `[x;y;z]`. Units are in meters.

Example: `[1000;100;500]`

Dependencies

To enable this argument, set the `ScattererSpecificationSource` property to `'Input port'`.

Data Types: `double`

`scatvel` — Velocities of scatterers
real-valued 3-by-N_s matrix

Velocities of scatterers, specified as a real-valued 3-by-N_s matrix. The matrix contains the (v_x,v_y,v_z) positions of scatterers. Each column of the matrix specifies a different scatterer and takes the form `[vx;vy;vz]` Units are in meters per second.

Example: `[1000;100;500]`

Dependencies

To enable this argument, set the `ScattererSpecificationSource` property to `'Input port'`.

Data Types: `double`

`scatcoef` — Scattering coefficients
complex-valued 1-by-$N_s$row vector

Scattering coefficients, specified as a complex-valued 1-by-$N_s$row vector. Each vector element specifies the scattering coefficient of the corresponding scatterer. Units are dimensionless.

Example: `[5+3*1i;4+1i;2]`

Dependencies

To enable this argument, set the `ScattererSpecificationSource` property to `'Input port'` and the `Polarization` property to `'None'`.

Data Types: `double`

`scatmat` — Scattering matrices

`[1 0;0 1]` | complex–valued 2-by-2-by-N_s array

Scattering matrices of the scatterers, specified as a complex–valued 2-by-2-by-N_s array where N_s is the number of scatterers. Each page of this array represents the scattering matrix of a scatterer. Each scattering matrix has the form `[s_hh s_hv;s_vh s_vv]`. For example, the component `s_hv` specifies the complex scattering response when the input signal is vertically polarized and the reflected signal is horizontally polarized. The other components are defined similarly. Units are in square meters.

Dependencies

To enable this property, set the `ScattererSpecificationSource` property to `'Input port'` and the `Polarization` property to `'Combined'` or `'Dual'`.

Data Types: `double`

`scataxes` — Scatterer orientation axes

real-valued 3-by-3-by-N_s array

Scatterer orientation axes, specified as a real-valued 3-by-3-by-N_s array where N_s is the number of scatterers. Each page of this array represents the orientation axes matrix of a scatterer. The columns of the matrix represent the *x*- ,*y*-, and *z*-axes of the scatterer. Units are dimensionless.

Dependencies

To enable this property, set the `ScattererSpecificationSource` property to `'Input port'` and the `Polarization` property to `'Combined'` or `'Dual'`.

Data Types: `double`

Output Arguments

Y — Received narrowband signal

complex-valued M-by-N_r matrix

Received narrowband signal, returned as an M-by-$\mathrm{N_r}$ complex-valued matrix. M is the number of samples in the signal, and N_r is the number of receiving array elements. Each column represents the signal received by the corresponding array element.

Example: `[1,1;j,1;0.5,0]`

Dependencies

To enable this argument, set the `Polarization` property to `'None'` or `'Combined'`.

Data Types: `double`

Complex Number Support: Yes

YH — Received narrowband H-polarization signal
complex-valued M-by-N_r matrix

Received narrowband H-polarization signal, returned as a complex-valued M-by-N_r matrix. M is the number of samples in the signal, and N_r is the number of receiving array elements. Each column represents the signal received by the corresponding array element.

Example: `[1,1;j,1;0.5,0]`

Dependencies

To enable this argument, set the `Polarization` property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

YV — Received narrowband V-polarization signal
complex-valued M-by-N_r matrix

Received narrowband V-polarization signal, returned as a complex-valued M-by-N_r matrix. M is the number of samples in the signal, and N_r is the number of receiving array elements. Each column represents the signal received by the corresponding array element.

Example: `[1,1;j,1;0.5,0]`

Dependencies

To enable this argument, set the `Polarization` property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

CR — Channel response
complex-valued N_t-by-N_r-by-N_c array

Channel response, returned as an N_t-by-N_r-by-N_c complex-valued array.

- N_t is the number of transmitting array elements.
- N_r is the number of receiving array elements.
- - When you specify `SimulateDirectPath` as `false`, $N_c = N_s$, the number of scatterers.
  - When you specify `SimulateDirectPath` as `true`, $N_c = N_s + 1$ to account for the direct path.

Each page of the array corresponds to the channel response matrix for a specific scatterer.

Dependencies

To enable this argument, set the `ChannelResponseOutputPort` property to `true` and the `Polarization` property to `'None'` or `'Combined'`.

Data Types: `double`
Complex Number Support: Yes

CR_HH — Channel response for H-input to H-output
complex-valued N_t-by-N_r-by-N_c array

Channel response from H-polarization input to H-polarization output returned as a complex-valued N_t-by-N_r-by-N_c array.

- N_t is the number of transmitting array elements.
- N_r is the number of receiving array elements.
- - When you specify `SimulateDirectPath` as `false`, $N_c = N_s$, the number of scatterers.
  - When you specify `SimulateDirectPath` as `true`, $N_c = N_s + 1$ to account for the direct path.

Each page of the array corresponds to the channel response matrix for a specific scatterer.

Dependencies

To enable this argument, set the `ChannelResponseOutputPort` property to `true` and the `Polarization` property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

`CR_HV` — Channel response for H-input to V-output

complex-valued N_t-by-N_r-by-N_c array

Channel response from H-polarization input to V-polarization output returned as a complex-valued N_t-by-N_r-by-N_c array.

- N_t is the number of transmitting array elements.
- N_r is the number of receiving array elements.
- - When you specify `SimulateDirectPath` as `false`, $N_c = N_s$, the number of scatterers.
  - When you specify `SimulateDirectPath` as `true`, $N_c = N_s + 1$ to account for the direct path.

Each page of the array corresponds to the channel response matrix for a specific scatterer.

Dependencies

To enable this argument, set the `ChannelResponseOutputPort` property to `true` and the `Polarization` property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

`CR_VH` — Channel response for V-input to H-output

complex-valued N_t-by-N_r-by-N_c array

Channel response from V-polarization input to H-polarization output returned as a complex-valued N_t-by-N_r-by-N_c array.

- N_t is the number of transmitting array elements.
- N_r is the number of receiving array elements.
- - When you specify `SimulateDirectPath` as `false`, $N_c = N_s$, the number of scatterers.
  - When you specify `SimulateDirectPath` as `true`, $N_c = N_s + 1$ to account for the direct path.

Each page of the array corresponds to the channel response matrix for a specific scatterer.

Dependencies

To enable this argument, set the `ChannelResponseOutputPort` property to `true` and the `Polarization` property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

CR_VV — Channel response for V-input to V-output
complex-valued N_t-by-N_r-by-N_c array

Channel response from V-polarization input to V-polarization output returned as a complex-valued N_t-by-N_r-by-N_c array.

- N_t is the number of transmitting array elements.
- N_r is the number of receiving array elements.
- - When you specify `SimulateDirectPath` as `false`, $N_c = N_s$, the number of scatterers.
  - When you specify `SimulateDirectPath` as `true`, $N_c = N_s + 1$ to account for the direct path.

Each page of the array corresponds to the channel response matrix for a specific scatterer.

Dependencies

To enable this argument, set the `ChannelResponseOutputPort` property to `true` and the `Polarization` property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

TAU — Path delays
1-by-N_s real-valued vector

Path delays, returned as a 1-by-N_C real-valued vector. Each element corresponds to the path time delay from the transmitting array phase center to a scatterer and then to the receiving array phase center.

- When you specify `SimulateDirectPath` as `false`, $N_c = N_s$, the number of scatterers.
- When you specify `SimulateDirectPath` as `true`, $N_c = N_s + 1$ to account for the direct path.

Dependencies

To enable this argument, set the `ChannelResponseOutputPort` property to `true`.

Data Types: `double`

Examples

Propagate Signals in MIMO Channel

Create a 30 GHz MIMO channel with random scatterers. The scenario contains a stationary 21-element transmitting ULA array and a stationary 15-element receiving ULA array. The transmitting antennas have cosine responses and the receiving antennas are isotropic. Element spacing for both arrays is less than one-half wavelength. The channel has 50 randomly generated static scatterers within a specified bounding box. The transmit array is located at [0;20;50] meters and the receive array is located at [200;10;10] meters. Compute the propagated signal through this channel. The sample rate for the signal is 10 MHz.

```
fc = 30e9;
c = physconst('LightSpeed');
lambda = c/fc;
fs = 10e6;
txarray = phased.ULA('Element',phased.CosineAntennaElement,...
    'NumElements',21,'ElementSpacing',0.45*lambda);
rxarray = phased.ULA('Element',phased.IsotropicAntennaElement,...
    'NumElements',15,'ElementSpacing',0.45*lambda);

channel = phased.ScatteringMIMOChannel('TransmitArray',txarray,...
    'ReceiveArray',rxarray,'PropagationSpeed',c,'CarrierFrequency',fc,...
    'SampleRate',fs,'TransmitArrayPosition',[0;20;50],...
    'ReceiveArrayPosition',[200;10;10],'NumScatterers',50,...
    'ScattererPositionBoundary',[10 180; -30 30; -30 30]);
```

Create a random data signal of ones and zeros for each transmitter.

```
x = randi(2,[100 21]) - 1;
```

Compute the received signals after propagating through the channel.

```
y = channel(x);
```

Propagate Signals in MIMO Channel from Moving Transmitter

Create a MIMO channel containing 3 fixed scatterers. The scenario contains a 21-element transmitting ULA array operating at 72 GHz, and a 15-element receiving ULA array. The transmitting elements have cosine response shapes and the receiving antennas are isotropic. Only the transmitting antenna is moving. Element spacing for both arrays is less than one-half wavelength. The transmitting array starts at (0,20,50) meters and moves towards the receiver at 2 m/s. The receiving array is located at (200,10,10) meters. Compute the propagated signal through this channel. The sample rate for the signal is 10 MHz.

```
fc = 72e9;
c = physconst('LightSpeed');
lambda = c/fc;
fs = 10e6;
txplatform = phased.Platform('MotionModel','Velocity','InitialPosition', ...
    [0;20;50],'Velocity',[2;0;0]);
txarray = phased.ULA('Element',phased.CosineAntennaElement, ...
    'NumElements',21,'ElementSpacing',0.45*lambda);
rxarray = phased.ULA('Element',phased.IsotropicAntennaElement, ...
    'NumElements',15,'ElementSpacing',0.45*lambda);
channel = phased.ScatteringMIMOChannel('TransmitArray',txarray, ...
    'ReceiveArray',rxarray,'PropagationSpeed',c,'CarrierFrequency',fc,...
    'SampleRate',fs,'TransmitArrayMotionSource','Input port', ...
    'ReceiveArrayMotionSource','Property','ReceiveArrayPosition',[200;10;10],...
    'ReceiveArrayOrientationAxes',rotz(180),...
    'ScattererSpecificationSource','Property','ScattererPosition', ...
    [75 100 120; -10 20 12; 5 -5 8],'ScattererCoefficient',[1i,2+3i,-1+1i]);
```

Move the platforms for two time steps at one second intervals. For each time instance:

- Create a random data signal of ones and zeros for each transmitter element.
- Move the transmitter and receiver. The orientations are fixed.
- Propagate the signals from transmitters to scatterers to receiver.

```
for k =1:2
    x = randi(2,[100 21]) - 1;
    [txpos,txvel] = txplatform(1);
    txaxes = eye(3);
    y = channel(x,txpos,txvel,txaxes);
end
```

Propagate Signals Through MIMO Channel to Moving Receiver

Create a MIMO channel containing 3 fixed scatterers. The scenario contains a 21-element transmitting ULA array and a 15-element receiving ULA array. Both arrays operate at 72 GHz. The transmitting elements have cosine response shapes and the receiving antennas are isotropic. Only the receiving antenna is moving. Element spacing for both arrays is less than one-half wavelength. The transmitting array is located at (0,20,50) meters. The receiving array starts at (200,10,10) meters and moves toward the transmitter at 2 m/s. Compute the propagated signal through this channel. The sample rate for the signal is 10 MHz.

```
fc = 72e9;
c = physconst('LightSpeed');
lambda = c/fc;
fs = 10e6;
rxplatform = phased.Platform('MotionModel','Velocity','InitialPosition',...
    [200;10;10],'Velocity',[-2;0;0]);
txarray = phased.ULA('Element',phased.CosineAntennaElement, ...
    'NumElements',21,'ElementSpacing',0.45*lambda);
rxarray = phased.ULA('Element',phased.IsotropicAntennaElement, ...
    'NumElements',15,'ElementSpacing',0.45*lambda);
channel = phased.ScatteringMIMOChannel('TransmitArray',txarray, ...
    'ReceiveArray',rxarray,'PropagationSpeed',c,'CarrierFrequency',fc, ...
    'SampleRate',fs,'TransmitArrayMotionSource','Property',...
    'TransmitArrayPosition',[0;20;50],'TransmitArrayOrientationAxes',eye(3,3), ...
    'ReceiveArrayMotionSource','Input port','ScattererSpecificationSource', ...
    'Property','ScattererPosition',[75 100 120; -10 20 12; 5 -5 8], ...
    'ScattererCoefficient',[1i,2+3i,-1+1i],'SpecifyAtmosphere',false);
```

Move the platforms for two time steps at one-second intervals. For each time instance:

- Create a random data signal of ones and zeros for each transmitter element.
- Move the transmitter and receiver. Fix the array orientations.
- Propagate the signals from transmitters to scatterers to receiver.

```
for k =1:2
    x = randi(2,[100 21]) - 1;
    [rxpos,rxvel] = rxplatform(1);
    rxaxes = rotz(45);
    y = channel(x,rxpos,rxvel,rxaxes);
end
```

Compute Propagated Signals Through MIMO Channel with Moving Scatterers

Create a MIMO channel containing 3 moving scatterers. The scenario contains a 21-element transmitting ULA array and a 15-element receiving ULA array. Both arrays operate at 72 GHz. The transmitting elements have cosine responses and the receiving antennas are isotropic. Element spacing for both arrays is less than one-half wavelength. The transmitting array is located at (0,20,50) meters. The receiving array is located at (200,10,10) meters. Compute the propagated signal through this channel. The sample rate for the signal is 10 MHz. Obtain the channel response matrix and time delays.

```
fc = 30e9;
c = physconst('LightSpeed');
lambda = c/fc;
fs = 10e6;
txarray = phased.ULA('Element',phased.CosineAntennaElement, ...
    'NumElements',21,'ElementSpacing',0.45*lambda);
rxarray = phased.ULA('Element',phased.IsotropicAntennaElement, ...
    'NumElements',15,'ElementSpacing',0.45*lambda);
channel = phased.ScatteringMIMOChannel('TransmitArray',txarray, ...
    'ReceiveArray',rxarray,'PropagationSpeed',c,'CarrierFrequency',fc, ...
    'SampleRate',fs,'TransmitArrayPosition',[0;20;50], ...
    'ReceiveArrayPosition',[200;10;10],'ScattererSpecificationSource','Input port', ...
    'ChannelResponseOutputPort',true);
```

Create a random data signal of ones and zeros for each transmitter.

```
x = randi(2,[100 21]) - 1;
```

Compute the received signals after propagating through the channel. Also return the channel matrix and delays.

```
scatpos = [75 100 120; -10 20 12; 5 -5 8];
scatvel = [0 0.5 0; -0.1 1.2 0.04; .05 -0.45 0.8];
scatcoef = [1i,2+3i,-1+1i];
[y,chmat,delays] = channel(x,scatpos,scatvel,scatcoef);
```

Display the dimensions of the channel matrix.

```
size(chmat)
```

```
ans = 1×3

    21    15     3
```

Display the time delays in microseconds.

```
delays*1e6
```

```
ans = 1×3

    0.7310    0.7196    0.6919
```

Propagate Polarized Signals in MIMO Channel

Create a MIMO channel at 30 GHz with a 16-element transmit array and a 64-element receive array. Assume the elements are short-dipole antennas and the arrays are uniform linear arrays. The transmit array is located at [0;0;50] meters.

The receive array has an initial position at [200;0;0] meters and is moving at a speed of [10;0;0] meters/second. There are 200 static scatterers randomly located on the *xy* plane within a square centered at [200;0;0] and with a side length of 100 meters.

Use the channel to compute the propagated polarized signal. Assume the sample rate for the signal is 10 MHz and the frame length is 1000 samples. Collect 5 frames of received signal.

```
fc = 30e9;
c = 3e8;
lambda = c/fc;
fs = 10e6;
txarray = phased.ULA('Element',phased.ShortDipoleAntennaElement,...
    'NumElements',16,'ElementSpacing',lambda/2);
rxarray = phased.ULA('Element',phased.ShortDipoleAntennaElement,...
    'NumElements',64,'ElementSpacing',lambda/2);

Ns = 200;
scatpos = [100*rand(1,Ns) + 150; 100*rand(1,Ns) + 150; zeros(1,Ns)];
temp = randn(1,Ns) + 1i*randn(1,Ns);
scatcoef = repmat(eye(2),1,1,Ns).*permute(temp,[1 3 2]);
scatax = repmat(eye(3),1,1,Ns);

Nframesamp = 1000;
Tframe = Nframesamp/fs;
rxmobile = phased.Platform('InitialPosition',[200;0;0],...
    'Velocity',[10;0;0],'OrientationAxesOutputPort',true);

chan = phased.ScatteringMIMOChannel(...
    'TransmitArray',txarray,...
    'ReceiveArray',rxarray,...
    'PropagationSpeed',c,...
    'CarrierFrequency',fc,...
    'SampleRate',fs,...
    'Polarization','Dual',...
    'TransmitArrayPosition',[0;0;50],...
    'ReceiveArrayMotionSource','Input port',...
    'ScattererSpecificationSource','Property',...
    'ScattererPosition',scatpos,...
    'ScatteringMatrix',scatcoef,...
    'ScattererOrientationAxes',scatax);

xh = randi(2,[Nframesamp 16])-1;
xv = randi(2,[Nframesamp 16])-1;

for m = 1:5
    [rxpos,rxvel,rxax] = rxmobile(Tframe);
    [yh,yv] = chan(xh,xv,rxpos,rxvel,rxax);
end
```

Version History

Introduced in R2017a

phased.SphericalWavefrontArrayResponse

Package: phased

Array response to spherical wavefront

Description

The phased.SphericalWavefrontArrayResponse System object calculates the complex-valued response of a sensor array to a spherical wave.

To compute the response:

1. Create the phased.SphericalWavefrontArrayResponse object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
response = phased.SphericalWaveformArrayResponse
response = phased.SphericalWaveformArrayResponse(Name,Value)
```

Description

The response = phased.SphericalWaveformArrayResponse creates a phased.SphericalWaveformArrayResponse System object response with default property values.

response = phased.SphericalWaveformArrayResponse(Name,Value) creates a phased.SphericalWaveformArrayResponse object with each property Name set to a specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN). Enclose each property name in single quotes.

Example: response = phased.SphericalWaveformArrayResponse('SensorArray',phased.URA([10,20],'ElementSpacing',[0.25,0.25]),'PropagationSpeed',physconst('LightSpeed')) creates a phased.SphericalWaveformArrayResponse System object for a 10-by-20 uniform rectangular array (URA) with element spacing set to 0.25 meters The propagation speed set to the speed of light.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

SensorArray — Sensor array
phased.ULA array with default array properties (default) | Phased Array System Toolbox array System object

Sensor array, specified as a Phased Array System Toolbox array System object.

Example: `phased.URA`

PropagationSpeed — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

IncludeElementResponse — Include individual element responses in the steering vector
`false` (default) | `true`

Option to include individual element responses in the steering vector, specified as `false` or `true`. If this property is set to `true`, the steering vector includes individual array element responses. If this property is set to `false`, the steering vector is computed assuming that the elements are isotropic, regardless of how the elements are specified. Set this property to true when using polarized signals.

When the array specified in the SensorArray property contains subarrays, the steering vector applies to subarrays. If SensorArray does not contain subarrays, the steering vector applies to the array elements.

Data Types: `logical`

WeightsInputPort — Enable weights input
`false` (default) | `true`

Enable weights input, specified as `false` or `true`. When `true`, use the object input argument `W` to specify weights. Weights are applied to individual array elements (or at the subarray level when subarrays are supported).

Data Types: `logical`

EnablePolarization — Enable simulation of polarization
`false` (default) | `true`

Enable polarization simulation, specified as `false` or `true`. Set this property to `true` to enable polarization and to `false` to ignore polarization. This property applies when the array specified in the `SensorArray` property is capable of simulating polarization and the `IncludeElementResponse` property of the array is set to `true`.

Data Types: `logical`

Usage

Syntax

```
RESP = response(FREQ,ANG,RNG)
RESP = response(FREQ,ANG,RNG,W)
```

Description

`RESP = response(FREQ,ANG,RNG)` returns the array response `RESP` at operating frequencies specified in `FREQ`, directions specified in `ANG`, and at range `RNG`.

`RESP = response(FREQ,ANG,RNG,W)` applies weights `W` to the sensor array. To enable this syntax, set the `WeightsInputPort` to `true`.

Input Arguments

`FREQ` — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`ANG` — Angles for computing response
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing response, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See “Azimuth and Elevation Angles”.

Example: `[45 60; 0 10]`

Data Types: `double`

RNG — Range
positive scalar | 1-by-*M* real-valued vector of positive values

Range, specified as a positive 1-by-*M* real-valued vector of positive values. If ANG has more than one column, RNG must be a scalar or have the same number of columns as ANG. Units are in meters.

Data Types: double

W — Array weights
1 (default) | complex length-*N* column vector | complex *N*-by-*L* matrix

Weights applied to sensor array elements, specified as a complex-valued, length-*N* column vector or a complex-valued, *N*-by-*L* matrix. *N* is the number of array elements and *L* is the number of frequencies specified in FREQ. If W is a vector, the same weight is applied at all frequencies. If W is a matrix, each column of W represents the weights used at the corresponding frequency specified in FREQ.

Dependencies

To enable this argument, set the WeightsInputPort to true.

Data Types: double
Complex Number Support: Yes

Output Arguments

RESP — Array response
complex-valued *M*-by-*L* matrix | struct

Array response, returned as a complex-valued *M*-by-*L* complex-valued matrix or struct containing complex values. The response depends on whether the EnablePolarization property is set to true or false.

- If the EnablePolarization property is set to false, the voltage response, RESP, has the dimensions *M*-by-*L*. *M* represents the number of angles specified in the input argument ANG while *L* represents the number of frequencies specified in FREQ.
- If the EnablePolarization property is set to true, the voltage response, RESP, is a MATLAB struct containing two complex fields, RESP.H and RESP.V. The RESP.H field represents the array's horizontal polarization response, while RESP.V represents the array's vertical polarization response. Each field has the dimensions *M*-by-*L*. *M* represents the number of angles specified in the input argument, ANG, while *L* represents the number of frequencies specified in FREQ.

Data Types: double
Complex Number Support: Yes

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named obj, use this syntax:

```
release(obj)
```

Common to All System Objects

step   Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Compute Spherical Wavefront Response for URA

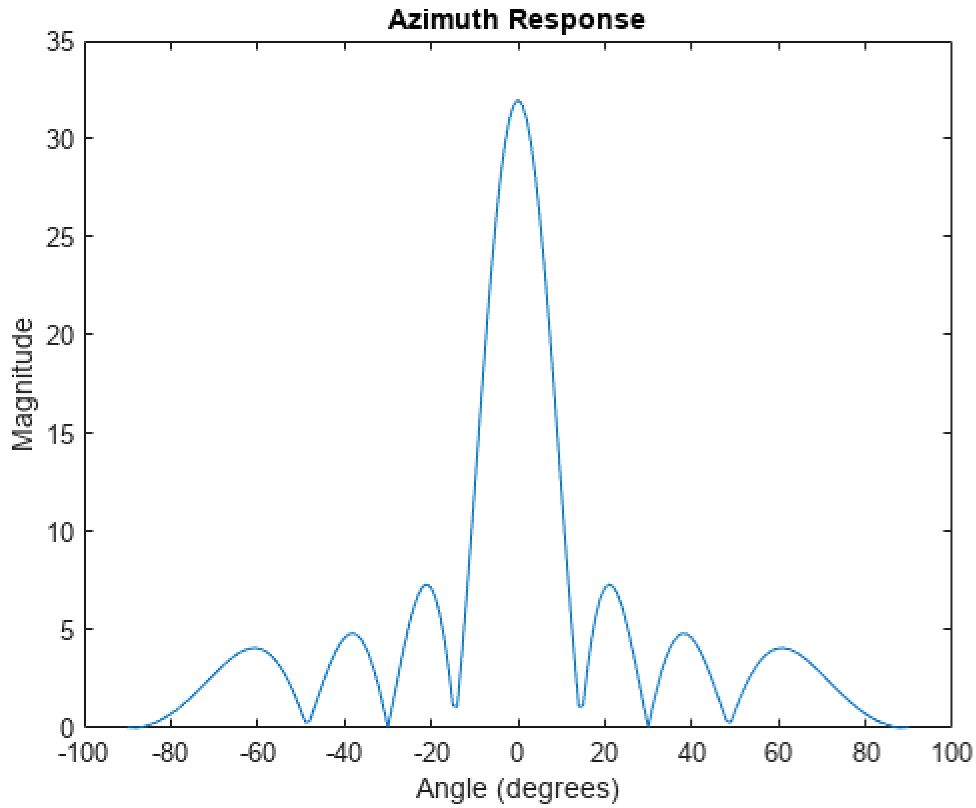
Calculate and plot the azimuth response of a 4-by-8 uniform rectangular array at directions between -90° and 90° in azimuth at a range of 1 km. Assume the array operating frequency is 300 MHz.

Create the array and the `phased.SphericalWavefrontArrayResponse System object™`.

```
array = phased.URA([4,8]);
response = phased.SphericalWavefrontArrayResponse( ...
    'SensorArray',array);
ang = -90:90;
rng = 1000.0;
fc = 300e6;
```

Compute and plot the response as a function of azimuth angle.

```
resp = response(fc,ang,rng);
plot(ang,abs(resp))
xlabel('Angle (degrees)')
ylabel('Magnitude')
title('Azimuth Response')
```



Compute Spherical Wavefront Response for Weighted ULA

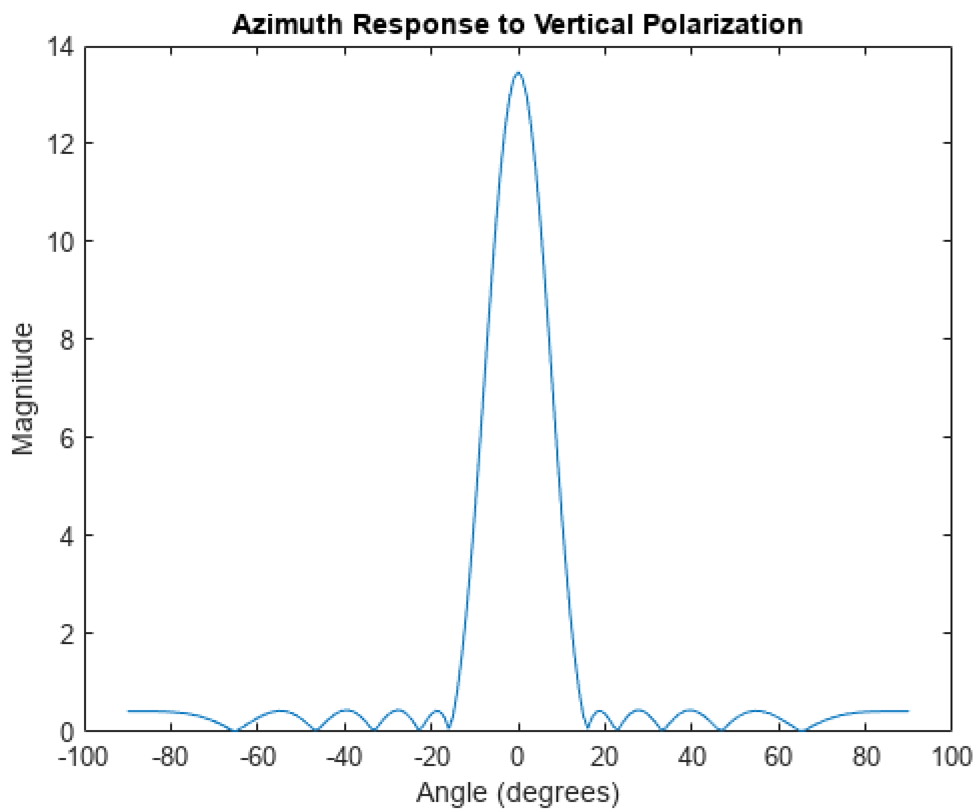
Calculate and plot the azimuth response of an 11-element uniform linear array at directions between -90° and 90° in azimuth at a range of 1 km. Create a polarized field using short-dipole antenna elements. Assume the array operating frequency is 300 MHz.

Create the ULA array and the `phased.SphericalWavefrontArrayResponse System object™`.

```
antenna = phased.ShortDipoleAntennaElement;
array = phased.ULA(11,'Element',antenna);
response = phased.SphericalWavefrontArrayResponse( ...
    'SensorArray',array,'EnablePolarization', ...
    true,'WeightsInputPort',true);
ang = -90:90;
rng = 1000.0;
fc = 300e6;
w = taylorwin(11);
```

Compute and plot the response as a function of azimuth angle.

```
resp = response(fc,ang,rng,w);
plot(ang,abs(resp.V))
xlabel('Angle (degrees)')
ylabel('Magnitude')
title('Azimuth Response to Vertical Polarization')
```



Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.ArrayResponse` | `phased.FocusedSteeringVector` | `phased.SteeringVector`

Topics
“Examine the Response of a Focused Phased Array”

phased.SteeringVector

Package: `phased`

Sensor array steering vector

Description

The `SteeringVector` System object creates steering vectors for a sensor array for multiple directions and frequencies.

To compute the steering vector for an array for specified directions and frequency

1. Create the `phased.SteeringVector` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
steervec = phased.SteeringVector
steervec = phased.SteeringVector(Name,Value)
```

Description

`steervec = phased.SteeringVector` creates a steering vector System object, `steervec`, with default property values.

`steervec = phased.SteeringVector(Name,Value)` creates a steering vector with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `steervec = phased.SteeringVector('SensorArray',phased.URA,'PropagationSpeed',physconst('LightSpeed'))` creates a steering vector object for a uniform rectangular array (URA) with default URA property values and sets the propagation speed to the speed of light.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`SensorArray` — Sensor array
`phased.ULA` array with default property values (default) | Phased Array System Toolbox array

Sensor array, specified as an array System object belonging to Phased Array System Toolbox. The sensor array can contain subarrays.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`IncludeElementResponse` — Include individual element responses in the steering vector
`false` (default) | `true`

Option to include individual element responses in the steering vector, specified as `false` or `true`. If this property is set to `true`, the steering vector includes individual array element responses. If this property is set to `false`, the steering vector is computed assuming that the elements are isotropic, regardless of how the elements are specified. Set this property to true when using polarized signals.

When the array specified in the SensorArray property contains subarrays, the steering vector applies to subarrays. If SensorArray does not contain subarrays, the steering vector applies to the array elements.

Data Types: `logical`

`NumPhaseShifterBits` — Number of phase shifter quantization bits
`0` (default) | nonnegative integer

Number of phase shifter quantization bits, specified as a nonnegative integer. This number of bits is used to quantize the phase shift component of the beamformer or steering vector weights. A value of zero indicates that no quantization is performed.

Data Types: `double`

`EnablePolarization` — Enable polarized fields
`false` (default) | `true`

Option to enable polarized fields, specified as `false` or `true`. Set this property to `true` to enable polarization. Set this property to `false` to ignore polarization. Enabling polarization requires that the sensor array specified in the SensorArray property can simulate polarization.

If you set this property to `false` for an array that actually supports polarization, then all polarization information is discarded. A combined pattern from the *H* and *V* polarization components is used at each element to compute the steering vector.

Data Types: `logical`

Usage

Syntax

```
SV = steervec(FREQ,ANG)
SV = steervec(FREQ,ANG,STEERANG)
SV = steervec(FREQ,ANG,STEERANG,WS)
```

Description

`SV = steervec(FREQ,ANG)` returns the steering vector, `SV`, pointing in the directions specified by `ANG` and for the operating frequencies specified in `FREQ`. The meaning of `SV` depends on the `IncludeElementResponse` property, as follows:

- If `IncludeElementResponse` is `true`, the components of `SV` include individual element responses.
- If `IncludeElementResponse` is `false`, the computation assumes that the elements are isotropic and `SV` does not include the individual element responses. If the array contains subarrays, `SV` is the array factor among the subarrays. The phase center of each subarray is at its geometric center. If SensorArray does not contain subarrays, `SV` is the array factor among the elements.

`SV = steervec(FREQ,ANG,STEERANG)` also specifies the subarray steering angle, `STEERANG`. To use this syntax, set the SensorArray property to an array type that contains subarrays and set the `IncludeElementResponse` to `true`. Arrays that contain subarrays are the `phased.PartitionedArray` and the `phased.ReplicatedSubarray`. In this case, set the `SubarraySteering` property of these arrays to either `'Phase'` or `'Time'`.

`SV = steervec(FREQ,ANG,STEERANG,WS)` also specifies `WS` as weights applied to each element within each subarray. To use this syntax, set the SensorArray property to an array that supports subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Input Arguments

ANG — Steering vector direction
`[0;0]` (default) | real-valued length-*M* vector | real-valued 2-by-*M* matrix

Steering vector directions, specified as a real-valued, length-*M* vector, or a real-valued 2-by-*M* matrix. *M* is the number of steering directions. When `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in space in the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, and the elevation angle must be between –90° and 90°. When `ANG` is a length-*M* vector, its values correspond to the azimuth angles of the steering vector direction with elevation angles set to zero. Angle units are in degrees.

Example: `[50.0,17.0,-24.5;0.4,4.0,23.9]`

Data Types: `single` | `double`

FREQ — Operating frequencies
1-by-*L* vector of positive values

Operating frequencies, specified as a 1-by-*L* vector of positive values. Units are in Hz.

Example: `[4100.0,4200.0]`

Data Types: `single` | `double`

STEERANG — Subarray steering direction
scalar | real-valued 2-by-1 vector

Subarray steering direction, specified as a scalar or a real-valued 2-by-1 vector. When `STEERANG` is a 2-by-1 vector, it specifies the subarray steering direction in the form `[azimuth;elevation]`. The azimuth angle must be between –180° and 180°, and the elevation angle must be between –90° and 90°. When `STEERANG` is a scalar, its value corresponds to the azimuth angle of the subarray steering direction with elevation angles set to zero. Angle units are in degrees.

Example: `[50.0;10.0]`

Data Types: `single` | `double`

WS — Subarray element weights
complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray element weights

Sensor Array	Subarray weights
`phased.ReplicatedSubarray`	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of `WS` specifies the weights for the corresponding subarray.
`phased.PartitionedArray`	Subarrays may not have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `Sensor` property to an array that contains subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

SV — Steering vector
complex-valued N-by-M-by-L array | structures

Steering vector, returned as a complex-valued *N*-by-*M*-by-*L* array or a structure containing arrays.

The form of the steering vector depends upon whether the EnablePolarization property is set to `true` or `false`.

- If EnablePolarization is set to `false`, the steering vector, `SV`, is an *N*-by-*M*-by-*L* array. The length of the first dimension, *N*, is the number of elements of the phased array. If SensorArray contains subarrays, *N* is the number of subarrays. The length of the second dimension, *M*, corresponding to the number of steering directions specified in the `ANG` argument. The length of the third dimension, *L*, is the number of frequencies specified in the `FREQ` argument.
- If EnablePolarization is set to `true`, `SV` is a MATLAB `struct` containing two fields, `SV.H` and `SV.V`. These two fields represent the horizontal (*H*) and vertical (*V*) polarization components of the steering vector. Each field is an *N*-by-*M*-by-*L* array. The length of the first dimension, *N*, is the number of elements of the phased array. If SensorArray contains subarrays, *N* is the number of subarrays. The length of the second dimension, *M*, corresponds to the number of steering directions specified in the `ANG` argument. The length of the third dimension, *L*, is the number of frequencies specified in the `FREQ` argument.

  Simulating polarization also requires that the sensor array specified in the SensorArray property can simulate polarization, and that the `IncludeElementResponse` property is set to `true`.

Data Types: `single` | `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Steering Vector for Uniform Linear Array

Calculate and display the steering vector for a 4-element uniform linear array in the direction of 30 degrees azimuth and 20 degrees elevation. Assume the array's operating frequency is 300 MHz.

```
array = phased.ULA('NumElements',4);
steervec = phased.SteeringVector('SensorArray',array);
fc = 3e8;
ang = [30; 20];
sv = steervec(fc,ang)
```

```
sv = 4×1 complex

  -0.6011 - 0.7992i
```

```
   0.7394 - 0.6732i
   0.7394 + 0.6732i
  -0.6011 + 0.7992i
```

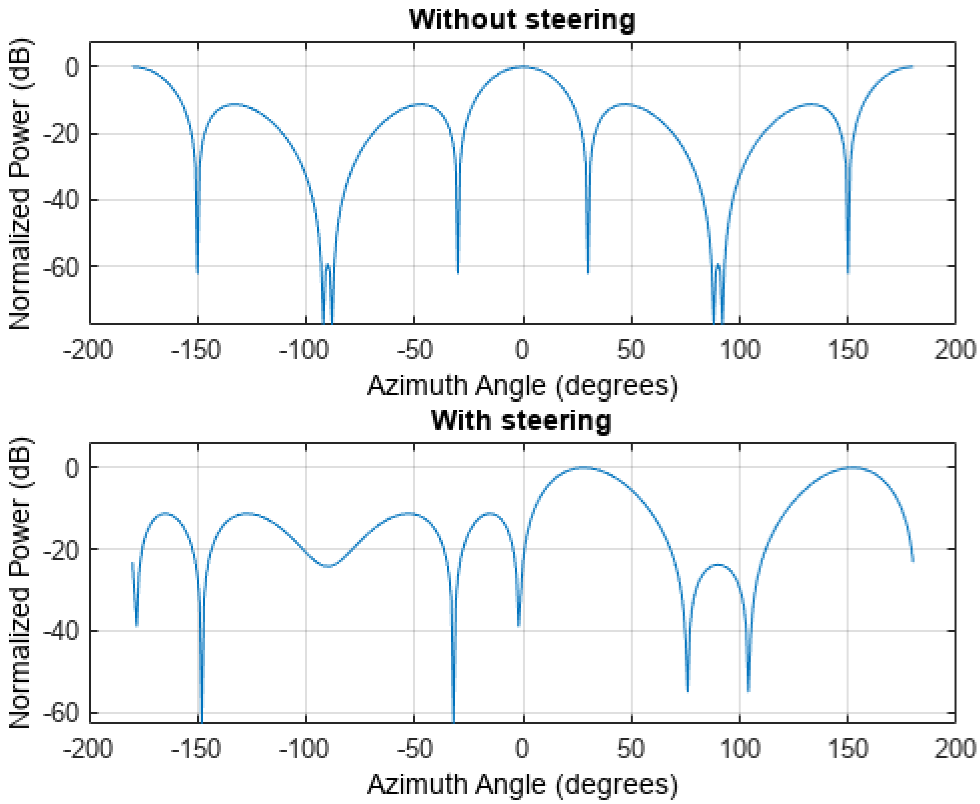
Beam Pattern with and Without Steering

Calculate the steering vector for a 4-element uniform linear array (ULA) in the direction of 30 degrees azimuth and 20 degrees elevation. Assume the array operating frequency is 300 MHz.

```
fc = 300e6;
c = physconst('LightSpeed');
array = phased.ULA('NumElements',4);
steervec = phased.SteeringVector('SensorArray',array);
sv = steervec(fc,[30;20]);
```

Plot the beam patterns for the uniform linear array when no steering vector is applied (steered broadside) and when a steering vector is applied.

```
subplot(211)
pattern(array,fc,-180:180,0,'CoordinateSystem','rectangular', ...
    'PropagationSpeed',c,'Type','powerdb')
title('Without steering')
subplot(212)
pattern(array,fc,-180:180,0,'CoordinateSystem','rectangular', ...
    'PropagationSpeed',c,'Type','powerdb','Weights',sv)
title('With steering')
```



Steering Vector for Uniform Linear Array

Calculate the steering vector for a uniform linear array in the direction of 30° azimuth and 20° elevation. Assume the array operates at 300 MHz.

```
array = phased.ULA('NumElements',2);
steeringvector = phased.SteeringVector('SensorArray',array);
fc = 300.0e6;
ang = [30;20];
sv = steeringvector(fc,ang);
```

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.ArrayGain` | `phased.ArrayResponse` | `phased.ElementDelay` | `uv2azel` | `phitheta2azel`

step

System object: `phased.SteeringVector`
Package: `phased`

Calculate steering vector

Syntax

```
SV = step(H,FREQ,ANG)
SV = step(H,FREQ,ANG,STEERANGLE)
SV = step(H,FREQ,ANG,WS)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`SV = step(H,FREQ,ANG)` returns the steering vector `SV` of the array for the directions specified in `ANG`. The operating frequencies are specified in `FREQ`. The meaning of `SV` depends on the `IncludeElementResponse` property of `H`, as follows:

- If `IncludeElementResponse` is `true`, `SV` includes the individual element responses.
- If `IncludeElementResponse` is `false`, the computation assumes the elements are isotropic and `SV` does not include the individual element responses. Furthermore, if the `SensorArray` property of `H` contains subarrays, `SV` is the array factor among the subarrays and the phase center of each subarray is at its geometric center. If `SensorArray` does not contain subarrays, `SV` is the array factor among the elements.

`SV = step(H,FREQ,ANG,STEERANGLE)` uses `STEERANGLE` as the subarray steering angle. This syntax is available when you configure `H` so that `H.Sensor` is an array that contains subarrays, `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`, and `H.IncludeElementResponse` is `true`.

`SV = step(H,FREQ,ANG,WS)` uses `WS` as weights applied to each element within each subarray. To use this syntax, set the `SensorArray` property to an array that supports subarrays and set the `SubarraySteering` property of the array to `'Custom'`, and `H.IncludeElementResponse` is `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Steering vector object.

FREQ

Operating frequencies in hertz. FREQ is a row vector of length L.

ANG

Directions in degrees. ANG can be either a 2-by-M matrix or a row vector of length M.

If ANG is a 2-by-M matrix, each column of the matrix specifies the direction in space in the form [azimuth; elevation]. The azimuth angle must be between –180 degrees and 180 degrees, and the elevation angle must be between –90 degrees and 90 degrees.

If ANG is a row vector of length M, each element specifies the direction azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

STEERANGLE

Subarray steering angle in degrees. STEERANGLE can be a length-2 column vector or a scalar.

If STEERANGLE is a length-2 vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180 degrees and 180 degrees, and the elevation angle must be between –90 degrees and 90 degrees.

If STEERANGLE is a scalar, it represents the azimuth angle. In this case, the elevation angle is assumed to be 0.

WS

Subarray element weights

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray Element Weights

Sensor Array	Subarray Weights
phased.ReplicatedSubarray	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of WS specifies the weights for the corresponding subarray.
phased.PartitionedArray	Subarrays cannot have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `SensorArray` property to an array that contains subarrays and set the `SubarraySteering` property of the array to `'Custom'`, and `H.IncludeElementResponse` is `true`.

Output Arguments

SV

Steering vector. The form of the steering vector depends upon whether the `EnablePolarization` property is set to `true` or `false`.

- If `EnablePolarization` is set to `false`, the steering vector, `SV`, has the dimensions N-by-M-by-L. The first dimension, N, is the number of elements of the phased array. If `H.SensorArray` contains subarrays, N is the number of subarrays. Each column of `SV` contains the steering vector of the array for the corresponding direction specified in `ANG`. Each of the L pages of `SV` contains the steering vectors of the array for the corresponding frequency specified in `FREQ`.

  If you set the `H.IncludeElementResponse` property to `true`, the steering vector includes the individual element responses. If you set the `H.IncludeElementResponse` property to `false`, the elements are assumed to be isotropic. Then, the steering vector does not include individual element responses.
- If `EnablePolarization` is set to `true`, `SV` is a MATLAB `struct` containing two fields, `SV.H` and `SV.V`. These fields represent the steering vector horizontal and vertical polarization components. Each field has the dimensions N-by-M-by-L. The first dimension, N, is the number of elements of the phased array. If `H.SensorArray` contains subarrays, N is the number of subarrays. Each column of `SV` contains the steering vector of the array for the corresponding direction specified in

ANG. Each of the *L* pages of SV contains the steering vectors of the array for the corresponding frequency specified in FREQ.

If you set EnablePolarization to false for an array that supports polarization, then all polarization information is discarded. The combined pattern from both H and V polarizations is used at each element to compute the steering vector.

Simulating polarization also requires that the sensor array specified in the SensorArray property can simulate polarization, and the IncludeElementResponse property is set to true.

Examples

Steering Vector for Uniform Linear Array

Calculate the steering vector for a uniform linear array in the direction of 30° azimuth and 20° elevation. Assume the array operates at 300 MHz.

```
array = phased.ULA('NumElements',2);
steeringvector = phased.SteeringVector('SensorArray',array);
fc = 300.0e6;
ang = [30;20];
sv = steeringvector(fc,ang);
```

See Also

uv2azel | phitheta2azel

phased.SteppedFMWaveform

Package: `phased`

Stepped FM pulse waveform

Description

The `SteppedFMWaveform` object creates a stepped FM pulse waveform.

To obtain waveform samples:

1. Define and set up your stepped FM pulse waveform. See "Construction" on page 1-1626.
2. Call `step` to generate the stepped FM pulse waveform samples according to the properties of `phased.SteppedFMWaveform`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

Construction

`sSFM = phased.SteppedFMWaveform` creates a stepped FM pulse waveform System object, `sSFM`. The object generates samples of a linearly stepped FM pulse waveform.

`sSFM = phased.SteppedFMWaveform(Name,Value)` creates a stepped FM pulse waveform object, `sSFM`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SampleRate`

Sample rate

Signal sample rate, specified as a positive scalar. Units are Hertz. The ratio of sample rate to pulse repetition frequency *(PRF)* must be a positive integer — each pulse must contain an integer number of samples.

Default: `1e6`

`DurationSpecification`

Method to set pulse duration

Method to set pulse duration (pulse width), specified as `'Pulse width'` or `'Duty cycle'`. This property determines how you set the pulse duration. When you set this property to `'Pulse width'`, then you set the pulse duration directly using the `PulseWidth` property. When you set this property to `'Duty cycle'`, you set the pulse duration from the values of the `PRF` and `DutyCycle` properties. The pulse width is equal to the duty cycle divided by the *PRF*.

Default: `'Pulse width'`

`PulseWidth`

Pulse width

Specify the length of each pulse (in seconds) as a positive scalar. The value must satisfy `PulseWidth <= 1./PRF`.

Default: `50e-6`

`DutyCycle`

Waveform duty cycle

Waveform duty cycle, specified as a scalar from 0 to 1, exclusive. This property applies when you set the `DurationSpecification` property to `'Duty cycle'`. The pulse width is the value of the `DutyCycle` property divided by the value of the `PRF` property.

Default: `0.5`

`PRF`

Pulse repetition frequency

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The*PRF* must satisfy these restrictions:

- The product of *PRF* and *PulseWidth* must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of `PRF` must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* using property settings alone or using property settings in conjunction with the `prfidx` input argument of the `step` method.

- When `PRFSelectionInputPort` is `false`, you set the *PRF* using properties only. You can
  - implement a constant *PRF* by specifying `PRF` as a positive real-valued scalar.
  - implement a staggered *PRF* by specifying `PRF` as a row vector with positive real-valued entries. Then, each call to the `step` method uses successive elements of this vector for the *PRF*. If the last element of the vector is reached, the process continues cyclically with the first element of the vector.
- When `PRFSelectionInputPort` is `true`, you can implement a selectable *PRF* by specifying `PRF` as a row vector with positive real-valued entries. But this time, when you execute the `step` method, select a *PRF* by passing an argument specifying an index into the *PRF* vector.

In all cases, the number of output samples is fixed when you set the `OutputFormat` property to `'Samples'`. When you use a varying *PRF* and set the `OutputFormat` property to `'Pulses'`, the number of samples can vary.

Default: `10e3`

PRFSelectionInputPort

Enable PRF selection input

Enable the PRF selection input, specified as `true` or `false`. When you set this property to `false`, the step method uses the values set in the `PRF` property. When you set this property to `true`, you pass an index argument into the `step` method to select a value from the PRF vector.

Default: `false`

FrequencyStep

Linear frequency step size

Specify the linear frequency step size (in hertz) as a positive scalar. The default value of this property corresponds to 20 kHz.

Default: `20e3`

NumSteps

Specify the number of frequency steps as a positive integer. When `NumSteps` is 1, the stepped FM waveform reduces to a rectangular waveform.

Default: `5`

FrequencyOffsetSource

Source of frequency offset

Source of frequency offset for the waveform, specified as `'Property'` or `'Input port'`.

- When you set this property to `'Property'`, the offset is determined by the value of the `FrequencyOffset` property.
- When you set this property to `'Input port'`, the `FrequencyOffset` is determined by the `freqoffset` input argument.

Default: `'Property'`

FrequencyOffset

Frequency offset

Frequency offset in Hz, specified as a scalar.

Dependencies

This property applies when you set the `FrequencyOffsetSource` property to `'Input port'`.

Default: `0` Hz

OutputFormat

Output signal format

Specify the format of the output signal as `'Pulses'` or `'Samples'`. When you set the `OutputFormat` property to `'Pulses'`, the output of the `step` method takes the form of multiple pulses specified by the value of the `NumPulses` property. The number of samples per pulse can vary if you change the pulse repetition frequency during the simulation.

When you set the `OutputFormat` property to `'Samples'`, the output of the `step` method is in the form of multiple samples. In this case, the number of output signal samples is the value of the `NumSamples` property and is fixed.

Default: `'Pulses'`

`NumSamples`

Number of samples in output

Specify the number of samples in the output of the `step` method as a positive integer. This property applies only when you set the `OutputFormat` property to `'Samples'`.

Default: `100`

`NumPulses`

Number of pulses in output

Specify the number of pulses in the output of the `step` method as a positive integer. This property applies only when you set the `OutputFormat` property to `'Pulses'`.

Default: `1`

`PRFOutputPort`

Set this property to `true` to output the PRF for the current pulse using a `step` method argument.

Dependencies

This property can be used only when the `OutputFormat` property is set to `'Pulses'`.

Default: `false`

`CoefficientsOutputPort`

Enable matched filter coefficients output port

Enable the matched filter coefficients output port, specified as `false` or `true`. When you set this property to `false`, the object does not provide the matched filter coefficients used during the simulation as an output. When you set this property to `true`, the object provides the matched filter coefficients used during the simulation as an output.

Default: `false`

Methods

bandwidth	Bandwidth of stepped FM pulse waveform
getMatchedFilter	Matched filter coefficients for waveform
plot	Plot stepped FM pulse waveform
reset	Reset state of stepped FM pulse waveform object
step	Samples of stepped FM pulse waveform

Common to All System Objects	
release	Allow System object property value changes

Examples

Plot Stepped-FM Waveform and Spectrum

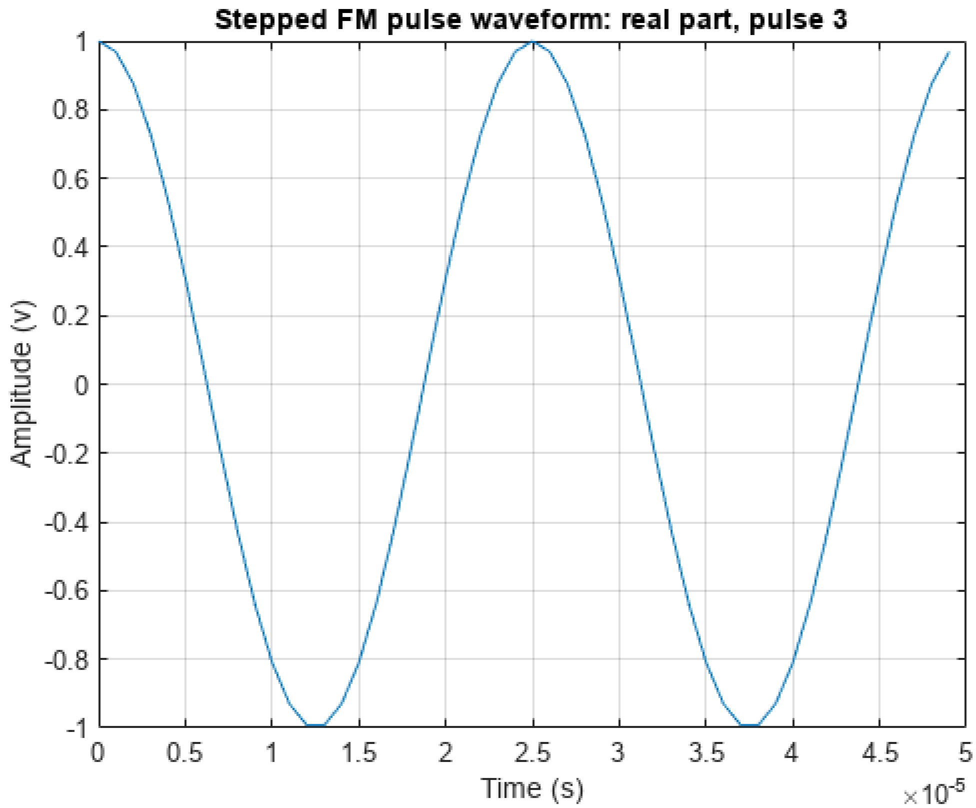
Create a stepped frequency pulse waveform object. Assume the default value, 1 MHz, for the sample rate. Then, plot the waveform.

Create the SteppedFMWaveform System object™ with 20 kHz frequency step size.

```
sSFM = phased.SteppedFMWaveform('NumSteps',3,'FrequencyStep',20e3);
fs = sSFM.SampleRate;
```

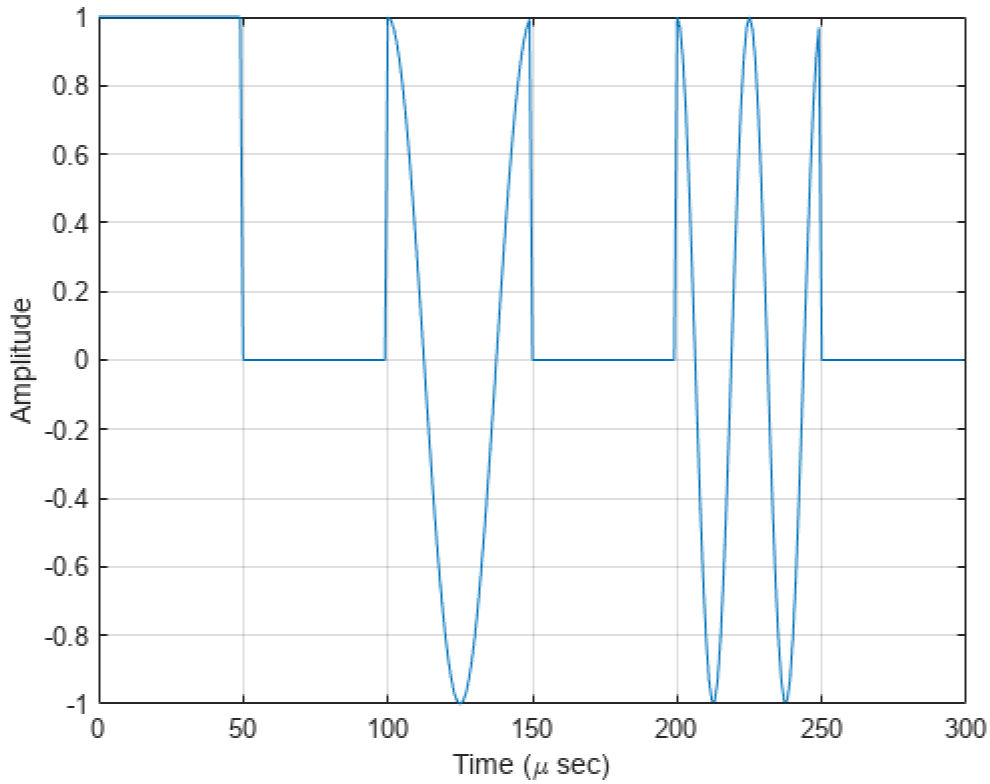
Plot the third pulse of the wave using the `phased.SteppedFMWaveform.plot` method. Pass in the pulse number using the `'PulseIdx'` name-value pair.

```
plot(sSFM,'PulseIdx',3);
```



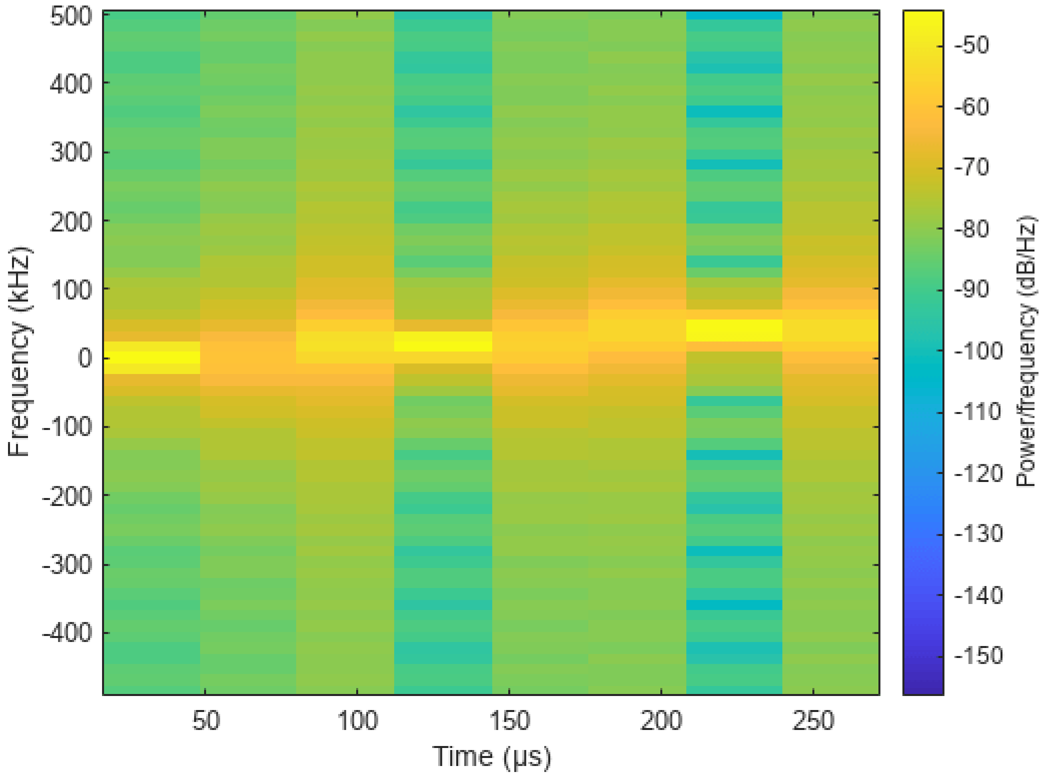
Alternatively, call the `step` method three times to obtain three pulses. Collect the three pulses in a single time series. Then plot the waveform using the `plot` function. You can see the full duty cycles of the pulses.

```
wavfull = [];
wav = step(sSFM);
wavfull = [wavfull;wav];
wav = step(sSFM);
wavfull = [wavfull;wav];
wav = step(sSFM);
wavfull = [wavfull;wav];
nsamps = size(wavfull,1);
t = [0:(nsamps-1)]/fs*1e6;
plot(t,real(wavfull))
xlabel('Time (\mu sec)')
ylabel('Amplitude')
grid
```



Plot the spectrum using the `spectrogram` function. Assume an fft of 64 samples and a 50% overlap. Window the signal with a hamming function.

```
nfft1 = 64;
nov = floor(0.5*nfft1);
spectrogram(wavfull,hamming(nfft1),nov,nfft1,fs,'centered','yaxis')
```



Apply Frequency Offset to Stepped FM Waveform

Apply a frequency offset to a stepped FM (SFM) pulse waveform. Plot the frequency spectrum of the waveform with and without a frequency offset applied.

Create an SFM waveform object which is configured to set the frequency offset from an input when the object is executed.

```
fs = 1e6;
sSFM = phased.SteppedFMWaveform('SampleRate',fs,'NumSteps',2, ...
    'FrequencyStep',20e3,'NumPulses',2,'FrequencyOffsetSource','Input port');
```

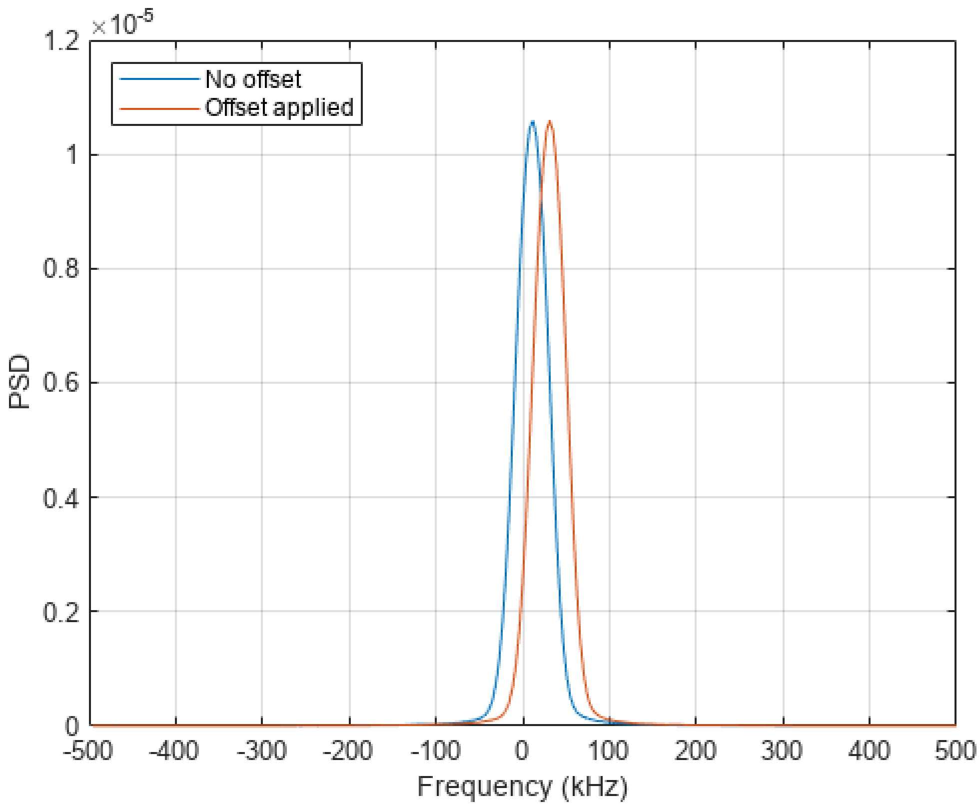
Execute the object two times. First set the frequency offset set to 0 Hz, and then to 2e4 Hz.

```
sfmwav = sSFM(0);
sfmwav_foffset = sSFM(2e4);
```

Plot the frequency spectrum of the complex signals. The frequency offset signal is shifted to the right.

```
[Pxx,f] = pwelch(sfmwav,[],[],[],fs,'centered');
[Pxx_offset,foffset] = pwelch(sfmwav_foffset,[],[],[],fs,'centered');
plot(f/1000,Pxx,foffset/1000,Pxx_offset)
ylabel('PSD');
xlabel('Frequency (kHz)');
```

```
legend({'No offset','Offset applied'},'Location','northwest');
grid on;
```



More About

Stepped FM Waveform

In a stepped FM waveform, a group of pulses together sweep a certain bandwidth. Each pulse in this group occupies a given center frequency and these center frequencies are uniformly located within the total bandwidth.

Version History

Introduced in R2011a

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- The `plot` method is not supported.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.LinearFMWaveform` | `phased.RectangularWaveform` | `phased.PhaseCodedWaveform`

Topics

“Waveform Analysis Using the Ambiguity Function”

bandwidth

System object: phased.SteppedFMWaveform
Package: phased

Bandwidth of stepped FM pulse waveform

Syntax

```
BW = bandwidth(H)
```

Description

`BW = bandwidth(H)` returns the bandwidth (in hertz) of the pulses for the stepped FM pulse waveform `H`. If there are N frequency steps, the bandwidth equals N times the value of the `FrequencyStep` property. If there is no frequency stepping, the bandwidth equals the reciprocal of the pulse width.

Input Arguments

H

Stepped FM pulse waveform object.

Output Arguments

BW

Bandwidth of the pulses, in hertz.

Examples

Bandwidth of Stepped FM Waveform

Determine the bandwidth of a stepped FM waveform.

```
waveform = phased.SteppedFMWaveform;
bw = bandwidth(waveform)
```

```
bw = 100000
```

getMatchedFilter

System object: phased.SteppedFMWaveform
Package: phased

Matched filter coefficients for waveform

Syntax

```
Coeff = getMatchedFilter(H)
Coeff = getMatchedFilter(H,'FrequencyOffset',FOFFSET)
```

Description

`Coeff = getMatchedFilter(H)` returns the matched filter coefficients for the stepped FM waveform object `H`. `Coeff` is a matrix whose columns correspond to the different frequency pulses in the stepped FM waveform.

`Coeff = getMatchedFilter(H,'FrequencyOffset',FOFFSET)` adds a frequency offset when matched filter coefficients are generated. `FOFFSET` must be a scalar. This option is available when you set the `FrequencyOffsetSource` property to `'Input port'` for the input object, `H`.

Examples

Matched Filter Coefficients for Stepped FM Pulse

Get the matched filter coefficients for a stepped FM pulse waveform.

```
waveform = phased.SteppedFMWaveform(...
    'NumSteps',3,'FrequencyStep',2e4,...
    'OutputFormat','Pulses','NumPulses',3);
coeff = getMatchedFilter(waveform);
```

Show the first four coefficients for each step.

```
coeff(1:4,:)
```

```
ans = 4×3 complex

   1.0000 + 0.0000i   0.9921 + 0.1253i   0.9686 + 0.2487i
   1.0000 + 0.0000i   0.9686 + 0.2487i   0.8763 + 0.4818i
   1.0000 + 0.0000i   0.9298 + 0.3681i   0.7290 + 0.6845i
   1.0000 + 0.0000i   0.8763 + 0.4818i   0.5358 + 0.8443i
```

plot

System object: `phased.SteppedFMWaveform`
Package: `phased`

Plot stepped FM pulse waveform

Syntax

```
plot(Hwav)
plot(Hwav,Name,Value)
plot(Hwav,Name,Value,LineSpec)
h = plot( ___ )
```

Description

`plot(Hwav)` plots the real part of the waveform specified by `Hwav`.

`plot(Hwav,Name,Value)` plots the waveform with additional options specified by one or more `Name,Value` pair arguments.

`plot(Hwav,Name,Value,LineSpec)` specifies the same line color, line style, or marker options as are available in the MATLAB `plot` function.

`h = plot( ___ )` returns the line handle in the figure.

Input Arguments

`Hwav`

Waveform object. This variable must be a scalar that represents a single waveform object.

`LineSpec`

Character vector to specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `PlotType` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Default: `'b'`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PlotType`

Specifies whether the function plots the real part, imaginary part, or both parts of the waveform. Valid values are `'real'`, `'imag'`, and `'complex'`.

Default: `'real'`

`PulseIdx`

Index of the pulse to plot. This value must be a scalar.

Default: `1`

`FrequencyOffset`

Frequency offset

Frequency offset in Hz, specified as a scalar.

Dependencies

This property applies when you set the `FrequencyOffsetSource` property to `'Input port'`.

Default: `0` Hz

Output Arguments

`h`

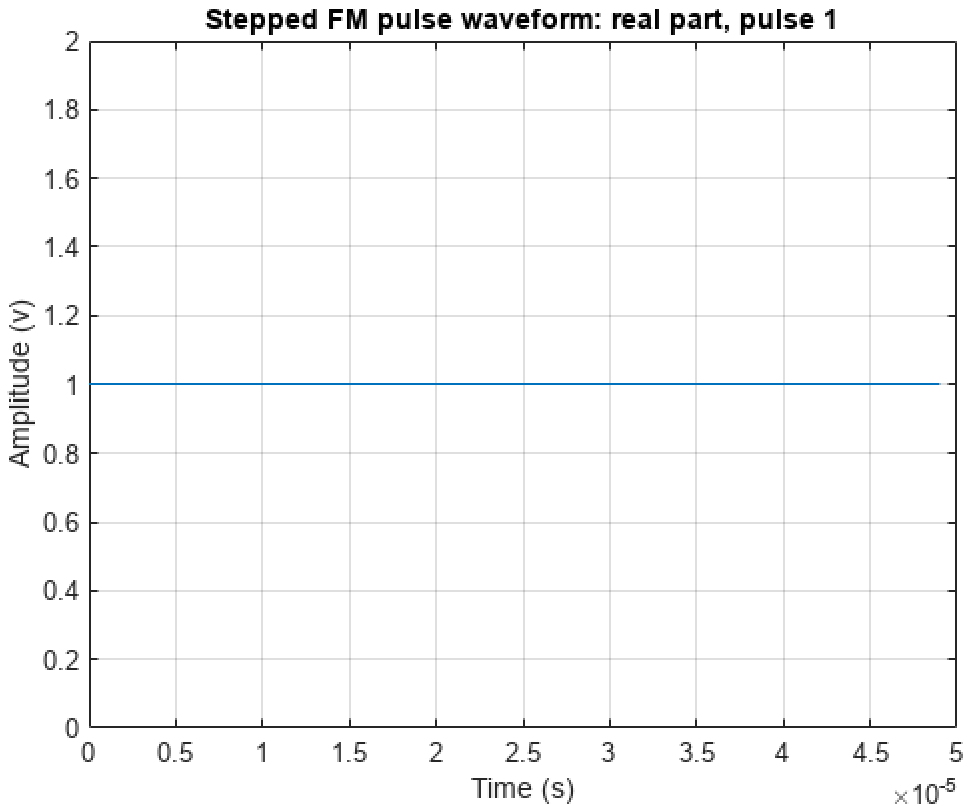
Handle to the line or lines in the figure. For a `PlotType` value of `'complex'`, `h` is a column vector. The first and second elements of this vector are the handles to the lines in the real and imaginary subplots, respectively.

Examples

Plot Stepped FM Waveform

Create and plot a stepped frequency pulse waveform.

```
waveform = phased.SteppedFMWaveform('NumSteps',3);
plot(waveform);
```


Stepped FM pulse waveform: real part, pulse 1
Amplitude (v)
Time (s)
×10^-5
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0.5
1.5
2.5
3
3.5
4
4.5
5

reset

System object: `phased.SteppedFMWaveform`
Package: `phased`

Reset state of stepped FM pulse waveform object

Syntax

```
reset(H)
```

Description

`reset(H)` resets the states of the `SteppedFMWaveform` object, `H`. Afterward, if the `PRF` property is a vector, the next call to `step` uses the first PRF value in the vector.

step

System object: `phased.SteppedFMWaveform`
Package: `phased`

Samples of stepped FM pulse waveform

Syntax

```
Y = step(sSFM)
Y = step(sSFM,prfidx)
Y = step(sRFM,freqoffset)
[Y,PRF] = step( ___ )
[Y,COEFF] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations. When the only argument to the `step` method is the System object itself, replace `y = step(obj)` by `y = obj()`.

`Y = step(sSFM)` returns samples of the stepped FM pulses in a column vector, `Y`. The output, `Y`, results from increasing the frequency of the preceding output by an amount specified by the `FrequencyStep` property. If the total frequency increase is larger than the value specified by the `SweepBandwidth` property, the samples of a rectangular pulse are returned.

`Y = step(sSFM,prfidx)`, uses the `prfidx` index to select the PRF from the predefined vector of values specified by in the `PRF` property. This syntax applies when you set the `PRFSelectionInputPort` property to `true`.

`Y = step(sRFM,freqoffset)`, uses the `freqoffset` to generate the waveform with an offset as specified at step time. Use this syntax for cases where the transmit pulse frequency needs to be dynamically updated. This syntax applies when you set the `FrequencyOffsetSource` property to `'Input port'`.

`[Y,PRF] = step( ___ )` also returns the current pulse repetition frequency, `PRF`. To enable this syntax, set the `PRFOutputPort` property to `true` and set the `OutputFormat` property to `'Pulses'`.

`[Y,COEFF] = step( ___ )` returns the matched filter coefficients, `COEFF`, for the current pulse. To enable this syntax, set `CoefficientsOutputPort` to `true`. `COEFF` is returned as either an N_Z-by-1 vector or an N_Z-by-M matrix.

- An N_Z-by-1 vector is returned when the object has `OutputFormat` set to `'Pulses'` and `NumPulses` is equal to 1. N_Z is the pulse width.
- An N_Z-by-M matrix is returned when either `OutputFormat` set to `'Pulses'` and `NumPulses` is greater than 1, or `OutputFormat` is set to `'Samples'`.

- When the object generates a constant pulse width waveform (`DurationSpecification` set to `'Pulse width'` or `'Duty cycle'` and `PRF` has one unique value), N_Z is the pulse width and M is the number of sub-pulses, `NumSteps`.
- When the object generates a varying pulse width waveform (`DurationSpecification` is set to `'Duty cycle'` and `PRF` has more than one unique value), N_Z is the maximum of the pulse width and M is the product of `NumSteps` and the number of unique PRFs.

You can combine optional input and output arguments when their enabling properties are set. Optional inputs and outputs must be listed in the same order as the order of the enabling properties. For example, `[Y,PRF,COEFF] = step(sRFM,prfidx,freqoffset)`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Create Stepped Frequency Pulse Waveform

Create a stepped frequency pulse waveform object with a frequency step of 40 kHz and four frequency steps.

```
waveform = phased.SteppedFMWaveform(...
    'NumSteps',4,'FrequencyStep',40e3,...
    'OutputFormat','Pulses','NumPulses',1);
fs = waveform.SampleRate;
```

Use the `waveform` method to obtain the pulses.

First, generate pulse 1.

```
pulse1 = waveform();
```

Then, generate pulse 2, incremented by the frequency step 40 kHz.

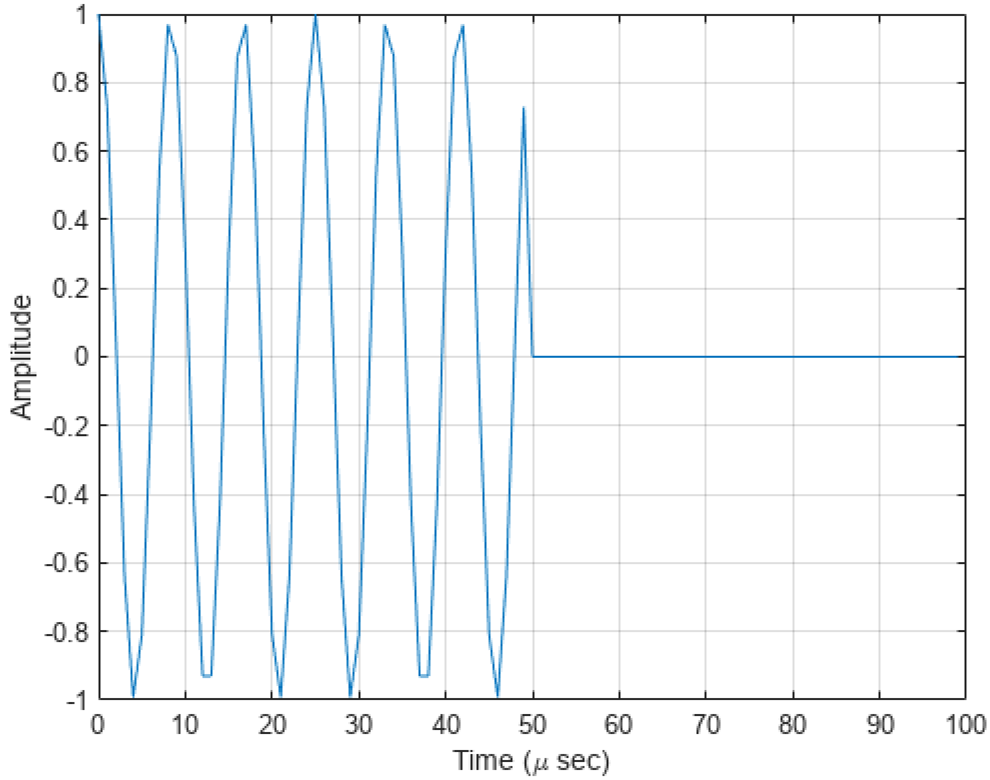
```
pulse2 = waveform();
```

Next, generate pulse 3, incremented by the frequency step 40 kHz.

```
pulse3 = waveform();
```

Finally, generate pulse 4, incremented by the frequency step 40 kHz.

```
pulse4 = waveform();
nsamps = size(pulse4,1);
t = [0:(nsamps-1)]/fs*1e6;
plot(t,real(pulse4))
xlabel('Time (\mu sec)')
ylabel('Amplitude')
grid
```



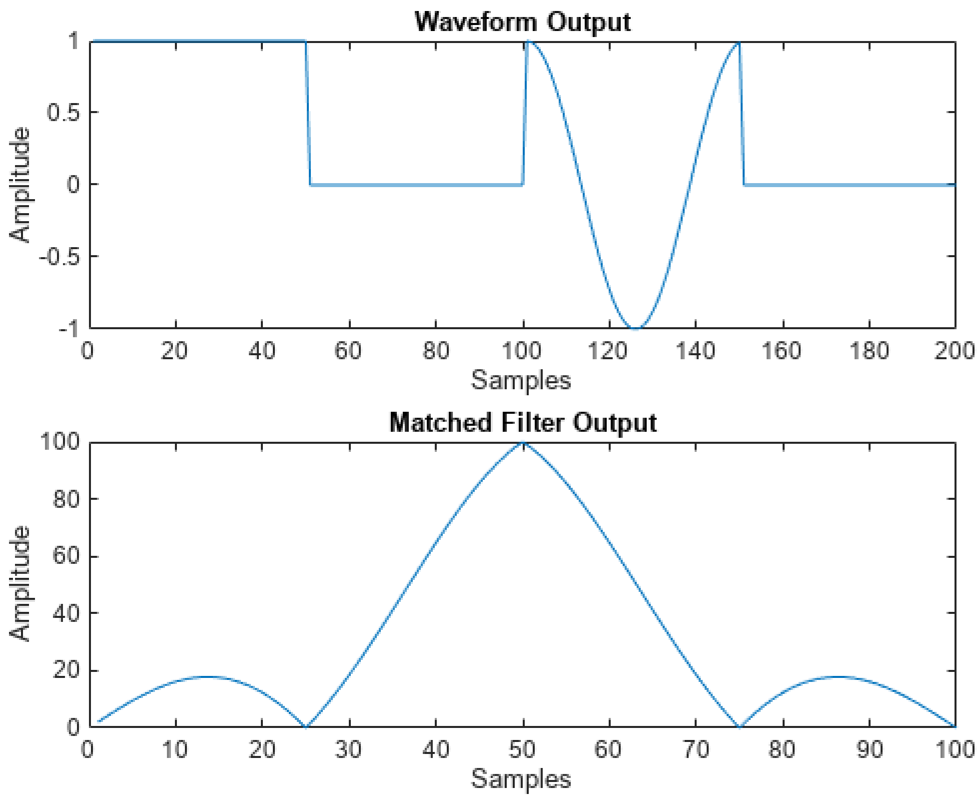
Generate Matched Filter Coefficients of Stepped FM Pulse Waveform

Generate output samples and matched filter coefficients of a stepped FM pulse waveform.

```
waveform = phased.SteppedFMWaveform('NumSteps',2,'NumPulses',1,...
    'CoefficientsOutputPort',true,'PRF',[1e4 1e4 2e4 2e4],...
    'DurationSpecification','Duty cycle','DutyCycle',0.5);
[wav1,coeff1] = waveform();
[wav2,coeff2] = waveform();
wav = [wav1 ; wav2];
```

Create a matched filter that applies the coefficients as an input argument. Use the coefficients when applying the matched filter to the waveform. Plot the waveform and matched filter outputs.

```
mf = phased.MatchedFilter('CoefficientsSource','Input port');
mfOut1 = mf(wav1,coeff1);
mfOut2 = mf(wav2,coeff2);
subplot(211),plot(real(wav));
xlabel('Samples'),ylabel('Amplitude'),title('Waveform Output');
subplot(212),plot(abs(mfOut1+mfOut2));
xlabel('Samples'),ylabel('Amplitude'),title('Matched Filter Output');
```



More About

Stepped FM Waveform

In a stepped FM waveform, a group of pulses together sweep a certain bandwidth. Each pulse in this group occupies a given center frequency and these center frequencies are uniformly located within the total bandwidth.

phased.StretchProcessor

Package: phased

Stretch processor for linear FM waveform

Description

The StretchProcessor object performs stretch processing on data from a linear FM waveform.

To perform stretch processing:

1. Define and set up your stretch processor. See "Construction" on page 1-1646.
2. Call step to perform stretch processing on input data according to the properties of phased.StretchProcessor. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.StretchProcessor creates a stretch processor System object, H. The object performs stretch processing on data from a linear FM waveform.

H = phased.StretchProcessor(Name,Value) creates a stretch processor object, H, with additional options specified by one or more Name,Value pair arguments. Name is a property name on page 1-1646, and Value is the corresponding value. Name must appear inside single quotes (''). You can specify several name-value pair arguments in any order as Name1,Value1,…,NameN,ValueN.

Properties

SampleRate

Sample rate

Signal sample rate, specified as a positive scalar. Units are Hertz. The ratio of sample rate to pulse repetition frequency *(PRF)* must be a positive integer — each pulse must contain an integer number of samples. This property can be specified as single or double precision.

Default: 1e6

PulseWidth

Pulse width

Specify the length of each pulse (in seconds) as a positive scalar. The value must satisfy PulseWidth <= 1./PRF. This property can be specified as single or double precision.

Default: 50e-6

`PRFSource`

Source of pulse repetition values

Source of the PRF values for the stretch processor, specified as `'Property'`, `'Auto'`, or `'Input port'`. When you set this property to `'Property'`, the PRF is determined by the value of the `PRF` property. When you set this property to `'Input port'`, the PRF is determined by an input argument to the `step` method at execution time. When you set this property to `'Auto'`, the PRF is computed from the number of rows in the input signal.

Default: `'Property'`

`PRF`

Pulse repetition frequency

Pulse repetition frequency (PRF) of the received signal, specified as a positive scalar. Units are in Hertz. This property can be specified as single or double precision.

Dependencies

To enable this property, set the `PRFSource` property to `'Property'`.

Default: `1`

`SweepSlope`

FM sweep slope

Specify the slope of the linear FM sweeping, in hertz per second, as a scalar.

Default: `2e9`

`SweepInterval`

Location of FM sweep interval

Specify the linear FM sweeping interval using the value `'Positive'` or `'Symmetric'`. If `SweepInterval` is `'Positive'`, the waveform sweeps in the interval between 0 and B, where B is the sweep bandwidth. If `SweepInterval` is `'Symmetric'`, the waveform sweeps in the interval between –B/2 and B/2. This property can be specified as single or double precision.

Default: `'Positive'`

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

`ReferenceRange`

Reference range of stretch processing

Specify the center of ranges of interest, in meters, as a positive scalar. The reference range must be within the unambiguous range of one pulse. This property can be specified as single or double precision. This property is tunable.

Default: `5000`

RangeSpan

Span of ranges of interest

Specify the length of the interval for ranges of interest, in meters, as a positive scalar. The range span is centered at the range value specified in the `ReferenceRange` property. This property can be specified as single or double precision.

Default: `500`

Methods

step	Perform stretch processing for linear FM waveform

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Detect a Target Using Stretch Processing

Use stretch processing to locate a target at a range of 4950 m.

Simulate the signal.

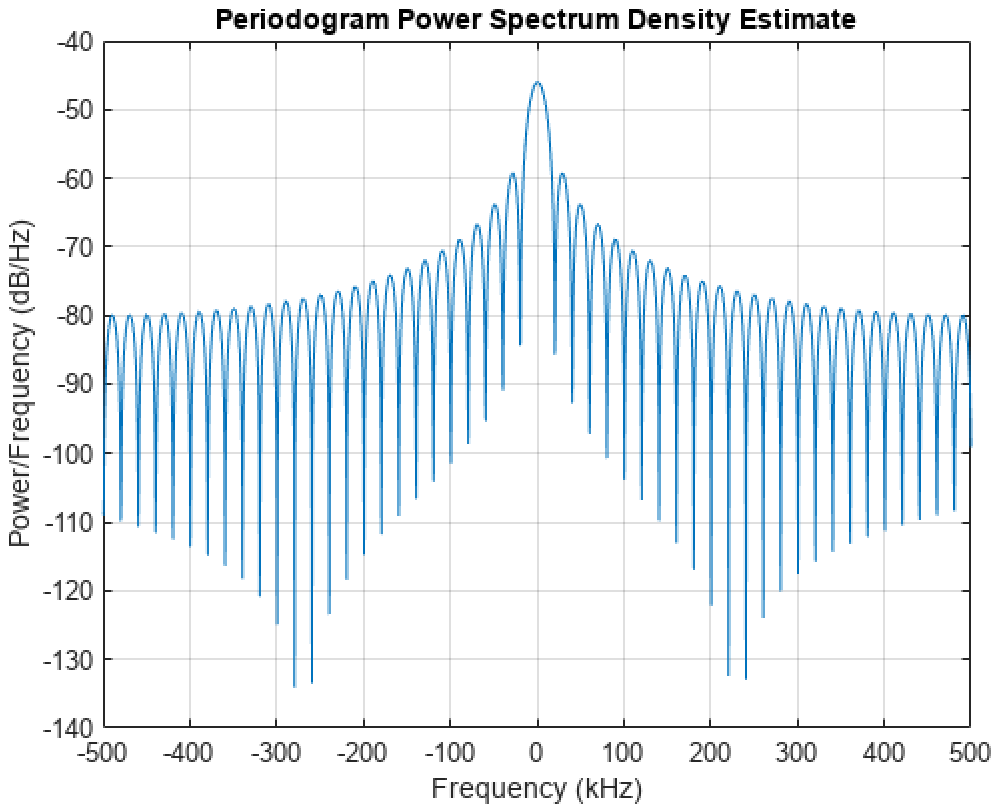
```
waveform = phased.LinearFMWaveform;
x = waveform();
c = physconst('LightSpeed');
rng = 4950.0;
num_samples = round(rng/(c/(2*waveform.SampleRate)));
x = circshift(x,num_samples);
```

Perform stretch processing.

```
stretchproc = getStretchProcessor(waveform,5000,200,c);
y = stretchproc(x);
```

Plot the spectrum of the resulting signal.

```
[Pxx,F] = periodogram(y,[],2048,stretchproc.SampleRate,'centered');
plot(F/1000,10*log10(Pxx))
grid
xlabel('Frequency (kHz)')
ylabel('Power/Frequency (dB/Hz)')
title('Periodogram Power Spectrum Density Estimate')
```



Detect the range.

```
[~,rngidx] = findpeaks(pow2db(Pxx/max(Pxx)),'MinPeakHeight',-5);
rngfreq = F(rngidx);
rng = stretchfreq2rng(rngfreq,stretchproc.SweepSlope,stretchproc.ReferenceRange,c)
```

```
rng = 4.9634e+03
```

Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2012a

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

phased.LinearFMWaveform | phased.MatchedFilter | stretchfreq2rng

Topics

Range Estimation Using Stretch Processing
"Stretch Processing"

step

System object: `phased.StretchProcessor`
Package: `phased`

Perform stretch processing for linear FM waveform

Syntax

```
Y = step(H,X)
Y = step(H,X,PRF)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` applies stretch processing along the first dimension of `X`. Each column of `X` represents one receiving pulse.

`Y = step(H,X,PRF)` uses `PRF` as the pulse repetition frequency. This syntax is available when the `PRFSource` property is `'Input port'`.

Input Arguments

H

Stretch processor object.

X

Input signal matrix. Each column represents one received pulse.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

PRF

Pulse repetition frequency specified as a positive scalar. To enable this argument, set the `PRFSource` property to `'Input port'`. Units are in Hertz.

Output Arguments

Y

Result of stretch processing. The dimensions of `Y` match the dimensions of `X`.

Examples

Detect a Target Using Stretch Processing

Use stretch processing to locate a target at a range of 4950 m.

Simulate the signal.

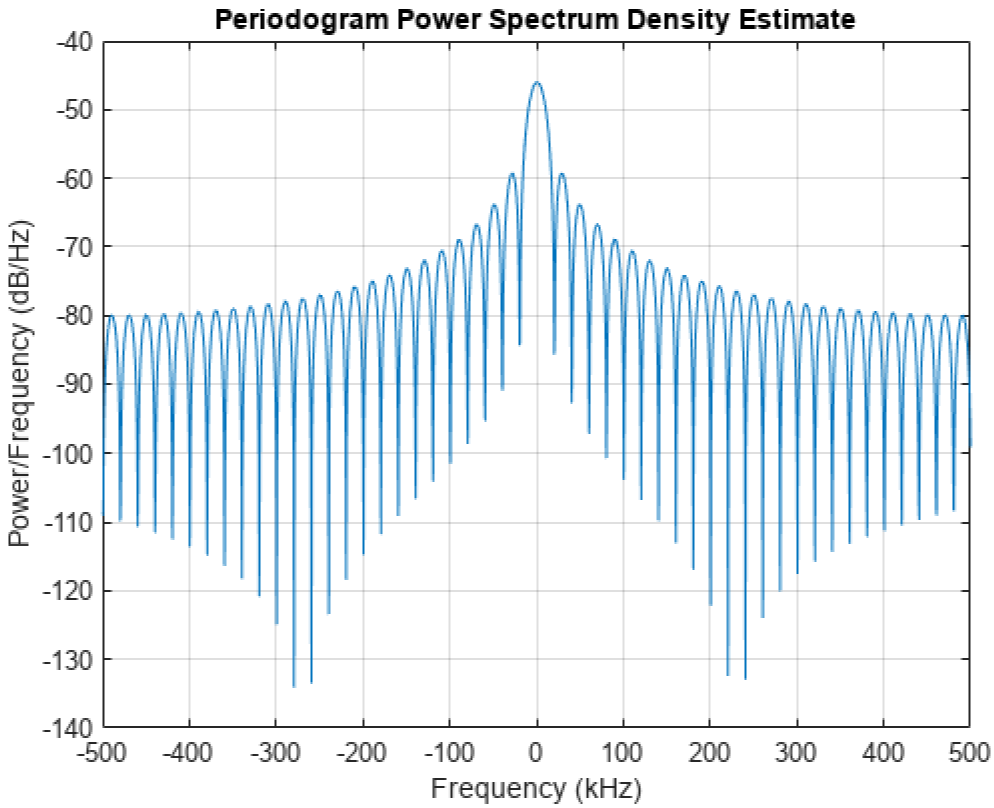
```
waveform = phased.LinearFMWaveform;
x = waveform();
c = physconst('LightSpeed');
rng = 4950.0;
num_samples = round(rng/(c/(2*waveform.SampleRate)));
x = circshift(x,num_samples);
```

Perform stretch processing.

```
stretchproc = getStretchProcessor(waveform,5000,200,c);
y = stretchproc(x);
```

Plot the spectrum of the resulting signal.

```
[Pxx,F] = periodogram(y,[],2048,stretchproc.SampleRate,'centered');
plot(F/1000,10*log10(Pxx))
grid
xlabel('Frequency (kHz)')
ylabel('Power/Frequency (dB/Hz)')
title('Periodogram Power Spectrum Density Estimate')
```



Detect the range.

```
[~,rngidx] = findpeaks(pow2db(Pxx/max(Pxx)),'MinPeakHeight',-5);
rngfreq = F(rngidx);
rng = stretchfreq2rng(rngfreq,stretchproc.SweepSlope,stretchproc.ReferenceRange,c)
```

```
rng = 4.9634e+03
```

See Also

`stretchfreq2rng`

Topics

Range Estimation Using Stretch Processing
"Stretch Processing"

phased.SubbandMVDRBeamformer

Package: phased

Wideband minimum-variance distortionless-response beamformer

Description

The phased.SubbandMVDRBeamformer System object implements a wideband minimum variance distortionless response beamformer (MVDR) based on the subband processing technique. This type of beamformer is also called a Capon beamformer.

To beamform signals arriving at an array:

1. Create the phased.SubbandMVDRBeamformer object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
beamformer = phased.SubbandMVDRBeamformer
beamformer = phased.SubbandMVDRBeamformer(Name,Value)
```

Description

beamformer = phased.SubbandMVDRBeamformer creates a subband MVDR beamformer System object, beamformer. The object performs subband MVDR beamforming on the received signal.

beamformer = phased.SubbandMVDRBeamformer(Name,Value) creates a subband MVDR beamformer System object, beamformer, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: beamformer = phased.SubbandMVDRBeamformer('SensorArray',phased.URA('Size',[5 5]),'OperatingFrequency',500e6) sets the sensor array to a 5-by-5 uniform rectangular array (URA) with all other default URA property values. The beamformer has an operating frequency of 500 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

SensorArray — Sensor array
phased.ULA array with default property values (default) | Phased Array System Toolbox array

Sensor array, specified as an array System object belonging to Phased Array System Toolbox. The sensor array can contain subarrays.

Example: phased.URA

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed').

Example: 3e8

Data Types: single | double

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: 1e9

Data Types: single | double

SampleRate — Sample rate of signal
1e6 (default) | positive scalar

Sample rate of signal, specified as a positive scalar. Units are in Hz. The System object uses this quantity to calculate the propagation delay in units of samples.

Example: 1e6

Data Types: single | double

NumSubbands — Number of processing subbands
64 (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: 128

Data Types: double

DirectionSource — Source of beamforming direction
'Property' (default) | 'Input port'

Source of beamforming direction, specified as 'Property' or 'Input port'. Specify whether the beamforming direction comes from the Direction property of this object or from the input argument, ANG. Values of this property are:

'Property'	Specify the beamforming direction using the Direction property.

'Input port'	Specify the beamforming direction using the input argument, ANG.

Data Types: char

Direction — Beamforming directions
[0;0] (default) | real-valued 2-by-1 vector | real-valued 2-by-*L* matrix

Beamforming directions, specified as a real-valued 2-by-1 vector or a real-valued 2-by-*L* matrix. For a matrix, each column specifies a different beamforming direction. Each column has the form [AzimuthAngle;ElevationAngle]. Azimuth angles must lie between –180° and 180° and elevation angles must lie between –90° and 90°. All angles are defined with respect to the local coordinate system of the array. Units are in degrees.

Example: [40;30]

Dependencies

To enable this property, set the DirectionSource property to 'Property'.

Data Types: single | double

DiagonalLoadingFactor — Diagonal loading factor
0 (default) | nonnegative scalar

Diagonal loading factor, specified as a nonnegative scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample size is small. A small sample size can lead to an inaccurate estimate of the covariance matrix. Diagonal loading also provides robustness due to steering vector errors. The diagonal loading technique adds a positive scalar multiple of the identity matrix to the sample covariance matrix.

Tunable: Yes

Data Types: single | double

TrainingInputPort — Enable training data input
false (default) | true

Enable training data input, specified as false or true. When you set this property to true, use the training data input argument, XT, when running the object. Set this property to false to use the input data, X, as the training data.

Data Types: logical

WeightsOutputPort — Enable beamforming weights output
false (default) | true

Enable the output of beamforming weights, specified as false or true. To obtain the beamforming weights, set this property to true and use the corresponding output argument, W. If you do not want to obtain the weights, set this property to false.

Data Types: logical

SubbandsOutputPort — Option to enable output of subband center frequencies
false (default) | true

Option to enable output of subband center frequencies, specified as either `true` or `false`. To obtain the subband center frequencies, set this property to `true` and use the corresponding output argument `FREQS` when calling the object.

Data Types: `logical`

Usage

Syntax

```
Y = beamformer(X)
Y = beamformer(X,XT)
Y = beamformer(X,ANG)
[Y,W] = beamformer(___)
[Y,FREQS] = beamformer(___)
[Y,W,FREQS] = beamformer(X,XT,ANG)
```

Description

`Y = beamformer(X)` performs wideband MVDR beamforming on the input, `X`, and returns the beamformed output in `Y`. This syntax uses `X` for training samples to calculate the beamforming weights. Use the Direction property to specify the beamforming direction.

`Y = beamformer(X,XT)` uses `XT` for training samples to calculate the beamforming weights.

`Y = beamformer(X,ANG)` uses `ANG` as the beamforming direction. This syntax applies when you set the DirectionSource property to `'Input port'`.

`[Y,W] = beamformer(___)` returns the beamforming weights, `W`. This syntax applies when you set the WeightsOutputPort property to `true`.

`[Y,FREQS] = beamformer(___)` returns the center frequencies of the subbands, `FREQS`. This syntax applies when you set the SubbandsOutputPort property to `true`.

You can combine optional input arguments when you set their enabling properties. Optional input arguments must be listed in the same order as their enabling properties. For example, `[Y,W,FREQS] = beamformer(X,XT,ANG)` is valid when you specify TrainingInputPort as `true` and set DirectionSource to `'Input port'`.

Input Arguments

X — Wideband input signal
M-by-*N* complex-valued matrix

Wideband input signal, specified as an M-by-N matrix, where N is the number of array elements. M is the number of samples in the data. If the sensor array consists of subarrays, N is then the number of subarrays.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

If you set the TrainingInputPort to `false`, then the object uses `X` as training data. In this case, the dimension *M* must be greater than *N*×*NB*. where *NB* is the number of subbands specified in the NumSubbands.

If you set TrainingInputPort to `true`, use the `XT` argument to supply training data. In this case, the dimension *M* can be any positive integer.

Example: `[1,1;j,1;0.5,0]`

Data Types: `single` | `double`
Complex Number Support: Yes

`XT` — Wideband training samples
P-by-*N* complex-valued matrix

Wideband training samples, specified as a *P*-by-*N* matrix where *N* is the number of elements. If the sensor array consists of subarrays, then *N* represents the number of subarrays.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

This argument applies when you set TrainingInputPort to `true`. The dimension *P* is the number of samples in the training data. *P* must be larger than *N*×*NB*, where *NB* is the number of subbands specified in the NumSubbands property.

Example: `FT = [1,1;j,1;0.5,0]`

Data Types: `single` | `double`
Complex Number Support: Yes

`ANG` — Beamforming direction
2-by-*L* real-valued matrix

Beamforming direction, specified as a *2*-by-*L* real-valued matrix, where *L* is the number of beamforming directions. This argument applies only when you set the DirectionSource property to `'Input port'`. Each column takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Example: `[40 30;0 10]`

Data Types: `single` | `double`

Output Arguments

`Y` — Beamformed output
M-by-*L* complex-valued matrix

Beamformed output, returned as an *M*-by-*L* complex-valued matrix. The quantity *M* is the number of signal samples and *L* is the number of beamforming directions specified in the `ANG` argument.

`W` — Beamforming weights
N-by-*K*-by-*L* complex-valued matrix

Beamforming weights, returned as an *N*-by-*K*-by-*L* complex-valued matrix. The quantity *N* is the number of sensor elements or subarrays and *K* is the number of subbands specified by the

NumSubbands property. The quantity *L* is the number of beamforming directions. Each column of `W` contains the narrowband beamforming weights used in the corresponding subband for the corresponding directions.

Dependencies

To return this output, set the WeightsOutputPort property to `true`.

Data Types: `single` | `double`

FREQS — Center frequencies of subbands
K-by-1 real-valued column vector

Center frequencies of subbands, returned as a *K*-by-1 real-valued column vector. The quantity *K* is the number of subbands specified by the NumSubbands property.

Dependencies

To return this output, set the SubbandsOutputPort property to `true`.

Data Types: `single` | `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Subband MVDR Beamforming of ULA

Apply subband MVDR beamforming to an underwater acoustic 11-element ULA. The incident angle of the signal is 10˚ azimuth and 30˚ elevation. The signal is an FM chirp having a bandwidth of 1 kHz. The speed of sound is 1500 m/s.

Simulate signal

```
array = phased.ULA('NumElements',11,'ElementSpacing',0.3);
fs = 2e3;
carrierFreq = 2000;
t = (0:1/fs:2)';
sig = chirp(t,0,2,fs/2);
c = 1500;
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c,...
    'SampleRate',fs,'ModulatedInput',true,...
    'CarrierFrequency',carrierFreq);
```

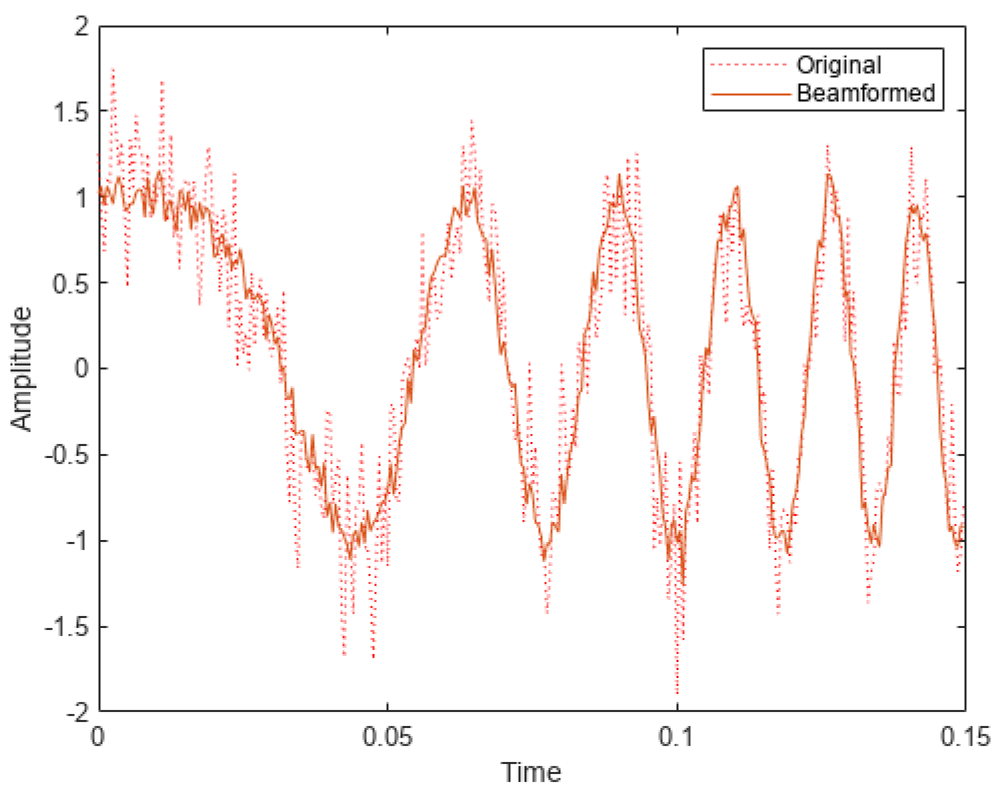
```
incidentAngle = [10;0];
sig1 = collector(sig,incidentAngle);
noise = 0.3*(randn(size(sig1)) + 1j*randn(size(sig1)));
rx = sig1 + noise;
```

Apply MVDR beamforming

```
beamformer = phased.SubbandMVDRBeamformer('SensorArray',array,...
    'Direction',incidentAngle,'OperatingFrequency',carrierFreq,...
    'PropagationSpeed',c,'SampleRate',fs,'TrainingInputPort',true, ...
    'SubbandsOutputPort',true,'WeightsOutputPort',true);
[y,w,subbandfreq] = beamformer(rx, noise);
```

Plot the signal that is input to the middle sensor (channel 6) vs the beamformer output.

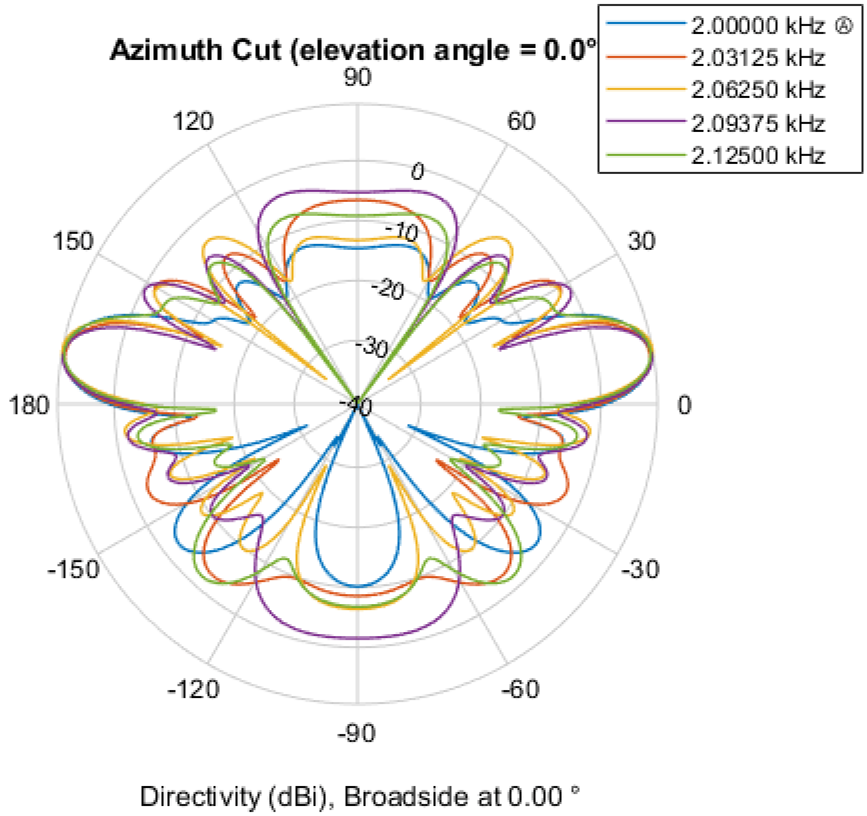
```
plot(t(1:300),real(rx(1:300,6)),'r:',t(1:300),real(y(1:300)))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed');
```



Plot array response

Plot the response pattern for five bands

```
pattern(array,subbandfreq(1:5).',-180:180,0,...
    'PropagationSpeed',c,'Weights',w(:,1:5));
```



Subband MVDR Beamforming of Array with Interference

Apply subband MVDR beamforming to an underwater acoustic 11-element ULA. Beamform the arriving signals to optimize the gain of a linear FM chirp signal arriving from 0 degrees azimuth and 0 degrees elevation. The signal has a bandwidth of 2.0 kHz. In addition, there unit amplitude 2.250 kHz interfering sine wave arriving from 28 degrees azimuth and 0 degrees elevation. Show how the MVDR beamformer nulls the interfering signal. Display the array pattern for several frequencies in the neighborhood of 2.250 kHz. The speed of sound is 1500 meters/sec.

Simulate Arriving Signal and Noise

```
array = phased.ULA('NumElements',11,'ElementSpacing',0.3);
fs = 2000;
carrierFreq = 2000;
t = (0:1/fs:2)';
sig = chirp(t,0,2,fs/2);
c = 1500;
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c,...
    'SampleRate',fs,'ModulatedInput',true,...
    'CarrierFrequency',carrierFreq);
incidentAngle = [0;0];
sig1 = collector(sig,incidentAngle);
noise = 0.3*(randn(size(sig1)) + 1j*randn(size(sig1)));
```

Simulate Interfering Signal

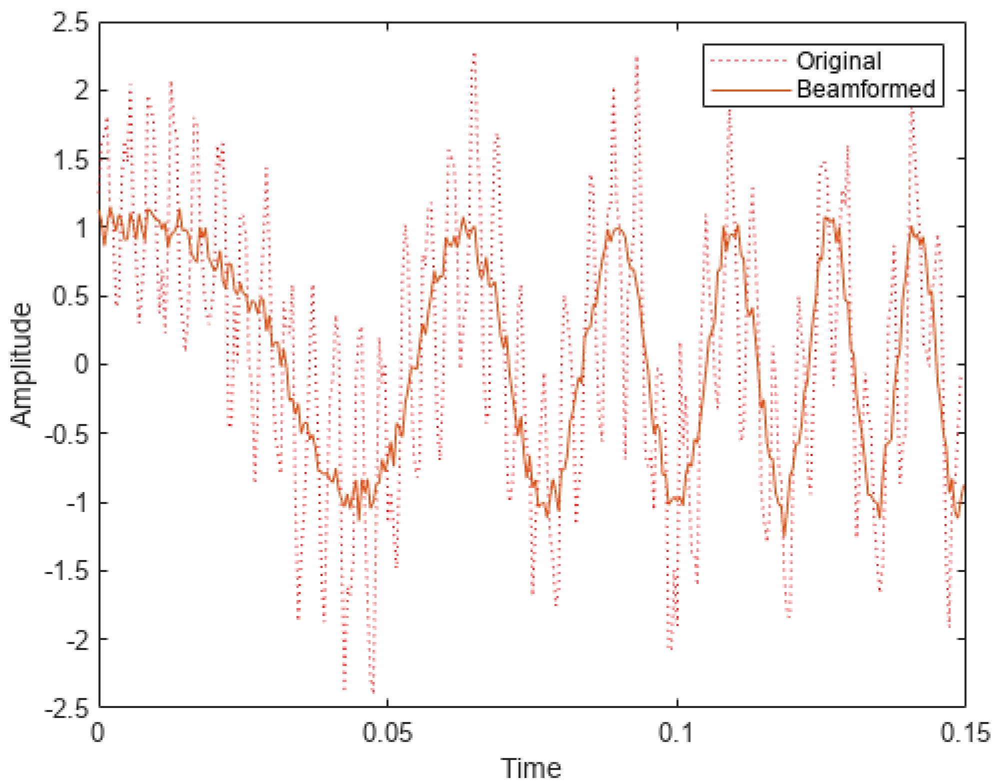
Combine both the desired and interfering signals.

```
fint = 250;
sigint = sin(2*pi*fint*t);
interfangle = [28;0];
sigint1 = collector(sigint,interfangle);
rx = sig1 + sigint1 + noise;
```

Apply MVDR Beamforming

Use the combined noise and interfering signal as training data.

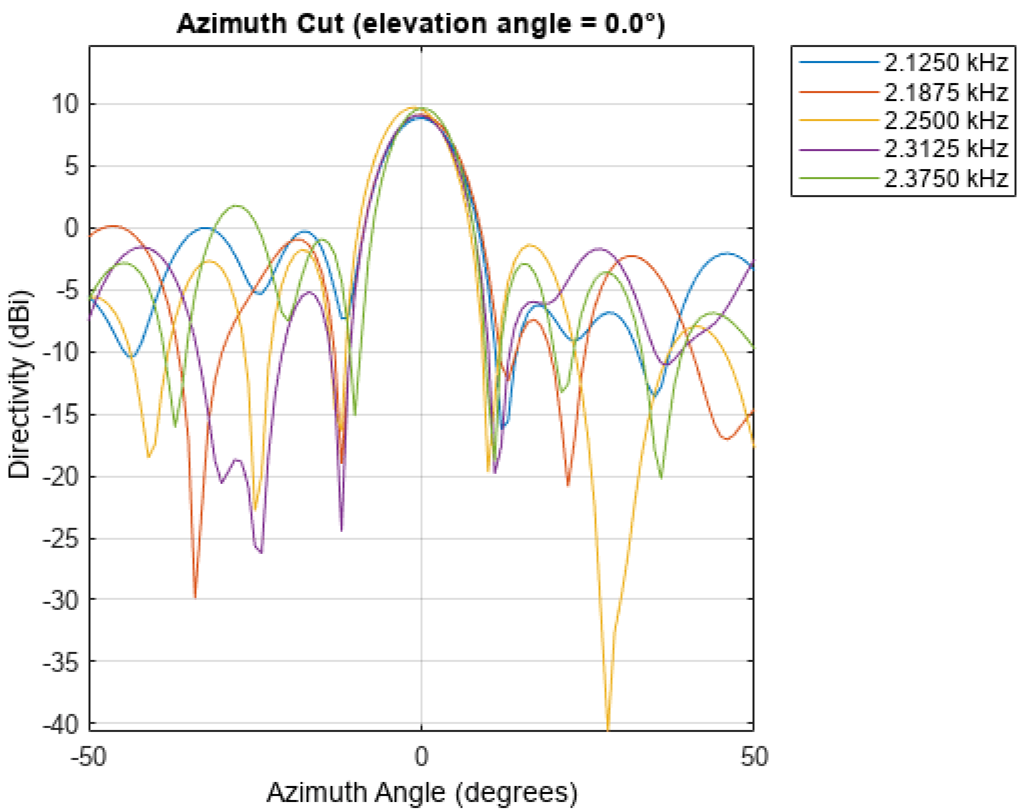
```
beamformer = phased.SubbandMVDRBeamformer('SensorArray',array,...
    'Direction',incidentAngle,'OperatingFrequency',carrierFreq,...
    'PropagationSpeed',c,'SampleRate',fs,'TrainingInputPort',true,...
    'NumSubbands',64,...
    'SubbandsOutputPort',true,'WeightsOutputPort',true);
[y,w,subbandfreq] = beamformer(rx,sigint1 + noise);
tidx = [1:300];
plot(t(tidx),real(rx(tidx,6)),'r:',t(tidx),real(y(tidx)))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Plot Array Response Showing Beampattern Null

Plot the response pattern for five bands near 2.250 kHz.

```
fdx = [5,7,9,11,13];
pattern(array,subbandfreq(fdx).',-50:50,0,...
    'PropagationSpeed',c,'Weights',w(:,fdx),...
    'CoordinateSystem','rectangular');
```



The beamformer places a null at 28 degrees for the subband containing 2.250 kHz.

More About

Diagonal Loading

Diagonal loading is a technique to improve beamformer robustness when stability issues arise from steering vector errors or finite sample size effects. This technique adds a positive real-valued multiple of the identity matrix to the correlation matrix of the received array data vector. You can apply diagonal loading using the DiagonalLoadingFactor property.

Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Subband Frequency Processing

Subband processing decomposes a wideband signal into multiple subbands and applies narrowband processing to the signal in each subband. The signals for all subbands are summed to form the output signal.

When using wideband frequency System objects or blocks, you specify the number of subbands, N_B, in which to decompose the wideband signal. Subband center frequencies and widths are automatically computed from the total bandwidth and number of subbands. The total frequency band is centered on the carrier or operating frequency, f_c. The overall bandwidth is given by the sample rate, f_s. Frequency subband widths are $\Delta f = f_s/N_B$. The center frequencies of the subbands are

$$f_m = \begin{cases} f_c - \frac{f_s}{2} + (m-1)\Delta f, & N_B \text{ even} \\ f_c - \frac{(N_B-1)f_s}{2N_B} + (m-1)\Delta f, & N_B \text{ odd} \end{cases}, \quad m = 1, \ldots, N_B$$

Some System objects let you obtain the subband center frequencies as output when you run the object. The returned subband frequencies are ordered consistently with the ordering of the discrete Fourier transform. Frequencies above the carrier appear first, followed by frequencies below the carrier.

Version History

Introduced in R2015b

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:.

- See “System Objects in MATLAB Code Generation” (MATLAB Coder)
- This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.MVDRBeamformer` | `phased.FrostBeamformer` | `phased.PhaseShiftBeamformer` | `phased.SubbandPhaseShiftBeamformer` | `phased.LCMVBeamformer` | `phased.WidebandCollector`

reset

System object: `phased.SubbandMVDRBeamformer`
Package: `phased`

Reset states of System object

Syntax

`reset(sMVDR)`

Description

`reset(sMVDR)` resets the internal state of the `phased.SubbandMVDRBeamformer` object, `sWBFS`. If the `SeedSource` property applies and has the value `'Property'`, then this method resets the state of the random number generator.

Input Arguments

`sMVDR` — Subband MVDR beamformer
System object

Subband MVDR beamformer, specified as a System object.

Example: `phased.SubbandMVDRBeamformer`

Version History

Introduced in R2015b

step

System object: `phased.SubbandMVDRBeamformer`
Package: `phased`

Wideband MVDR beamforming

Syntax

```
Y = step(sMVDR,X)
Y = step(sMVDR,X,XT)
Y = step(sMVDR,X,ang)
[Y,Wts] = step(sMVDR,___)
[Y,Freq] = step(sMVDR,___)
[Y,Wts,Freq] = step(sMVDR,X,XT,ang)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(sMVDR,X)` performs wideband MVDR beamforming on the input, `X`, and returns the beamformed output in `Y`. This syntax uses `X` for training samples to calculate the beamforming weights. Use the `Direction` property to specify the beamforming direction.

`Y = step(sMVDR,X,XT)` uses `XT` as the training samples to calculate the beamforming weights. This syntax applies only when you set the `TrainingInputPort` property to `true`. Use the `Direction` property to specify the beamforming direction.

`Y = step(sMVDR,X,ang)` uses `ang` as the beamforming direction. This syntax applies only when you set the `DirectionSource` property to `'Input port'`.

`[Y,Wts] = step(sMVDR,___)` returns the beamforming weights, `Wts`, when you set the `WeightsOutputPort` property to `true`.

`[Y,Freq] = step(sMVDR,___)` returns the center frequencies of the subbands, `Freq`, when you set the `SubbandsOutputPort` property to true. `Freq` is a length-*K* column vector where, *K* is the number of subbands specified in the `NumSubbands` property.

You can combine optional input arguments when you set their enabling properties. Optional input arguments must be listed in the same order as their enabling properties. For example, `[Y,Wts,Freq] = step(sMVDR,X,XT,ang)` is valid when you specify `TrainingInputPort` to `true` and specify `DirectionSource` to `'Input port'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`sMVDR` — Subband MVDR beamformer
System object

Subband MVDR beamformer, specified as a System object.

Example: `phased.SubbandMVDRBeamformer`

`X` — Wideband input field
M-by-*N* complex-valued matrix

Wideband input field, specified as an *M*-by-*N* matrix, where *N* is the number of array elements. If the sensor array consists of subarrays, *N* is then the number of subarrays. *M* is the number of samples in the data.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

If you set the `TrainingInputPort` to `false`, then `step` uses `X` as training data. In this case, the dimension *M* must be greater than $N \times NB$. where *NB* is the number of subbands specified in the `NumSubbands` property.

If you set `TrainingInputPort` to `true`, use the `XT` argument to supply training data. In this case, the dimension *M* can be any positive integer.

Example: `[1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

`XT` — Wideband training samples
P-by-*N* complex-valued matrix

Wideband training samples, specified as a *P*-by-*N* matrix where *N* is the number of elements. If the sensor array consists of subarrays, then *N* represents the number of subarrays.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

This argument applies when you set `TrainingInputPort` to `true`. The dimension *P* is the number of samples in the training data. *P* must be larger than $N \times NB$, where *NB* is the number of subbands specified in the `NumSubbands` property.

Example: `FT = [1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

`ang` — Beamforming direction
2-by-*L* real-valued matrix

Beamforming direction, specified as a 2-by-*L* real-valued matrix, where *L* is the number of beamforming directions. This argument applies only when you set the `DirectionSource` property to `'Input port'`. Each column takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Example: `F = [40 30; 0 10]`

Data Types: `double`

Output Arguments

`Y` — Beamformed output

M-by-*L* complex-valued matrix

Beamformed output, returned as an *M*-by-*L* complex-valued matrix. The quantity *M* is the number of signal samples and *L* is the number of beamforming directions specified in the `ang` argument.

`Wts` — Beamforming weights

N-by-*K*-by-*L* complex-valued matrix

Beamforming weights, returned as an *N*-by-*K*-by-*L* complex-valued matrix. The quantity *N* is the number of sensor elements or subarrays and *K* is the number of subbands specified by the `NumSubbands` property. The quantity *L* is the number of beamforming directions. Each column of `Wts` contains the narrowband beamforming weights used in the corresponding subband for the corresponding directions. This output applies only when you set the `WeightsOutputPort` property to `true`.

`Freq` — Center frequencies of subbands

K-by-1 real-valued column vector

Center frequencies of subbands, returned as a *K*-by-1 real-valued column vector. The quantity *K* is the number of subbands specified by the `NumSubbands` property. To return this output, set the `SubbandsOutputPort` property to `true`.

Examples

Subband MVDR Beamforming of ULA

Apply subband MVDR beamforming to an underwater acoustic 11-element ULA. The incident angle of the signal is 10˚ azimuth and 30˚ elevation. The signal is an FM chirp having a bandwidth of 1 kHz. The speed of sound is 1500 m/s.

Simulate signal

```
array = phased.ULA('NumElements',11,'ElementSpacing',0.3);
fs = 2e3;
carrierFreq = 2000;
t = (0:1/fs:2)';
sig = chirp(t,0,2,fs/2);
c = 1500;
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c,...
    'SampleRate',fs,'ModulatedInput',true,...
```

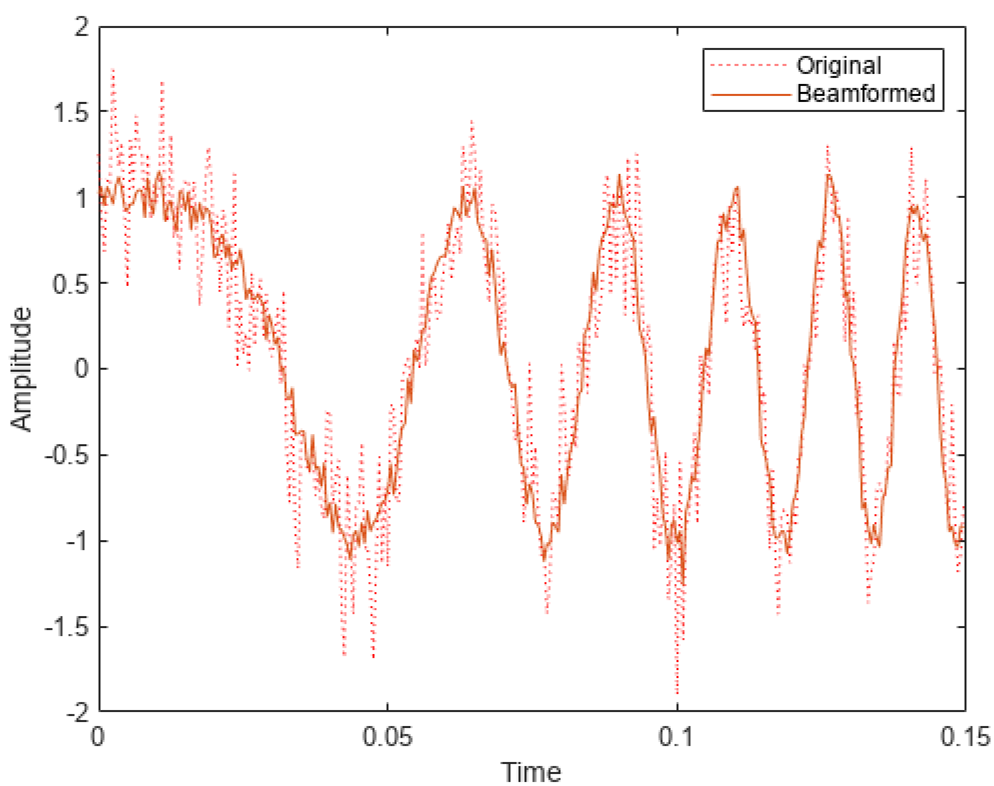
```
    'CarrierFrequency',carrierFreq);
incidentAngle = [10;0];
sig1 = collector(sig,incidentAngle);
noise = 0.3*(randn(size(sig1)) + 1j*randn(size(sig1)));
rx = sig1 + noise;
```

Apply MVDR beamforming

```
beamformer = phased.SubbandMVDRBeamformer('SensorArray',array,...
    'Direction',incidentAngle,'OperatingFrequency',carrierFreq,...
    'PropagationSpeed',c,'SampleRate',fs,'TrainingInputPort',true, ...
    'SubbandsOutputPort',true,'WeightsOutputPort',true);
[y,w,subbandfreq] = beamformer(rx, noise);
```

Plot the signal that is input to the middle sensor (channel 6) vs the beamformer output.

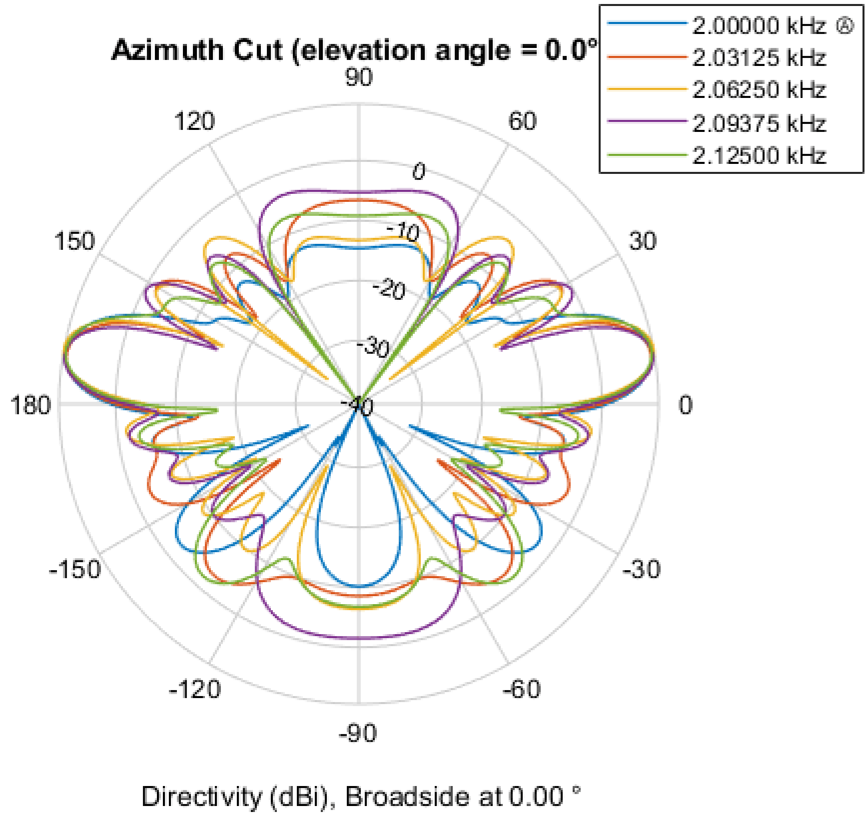
```
plot(t(1:300),real(rx(1:300,6)),'r:',t(1:300),real(y(1:300)))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed');
```



Plot array response

Plot the response pattern for five bands

```
pattern(array,subbandfreq(1:5).',-180:180,0,...
    'PropagationSpeed',c,'Weights',w(:,1:5));
```



Subband MVDR Beamforming of Array with Interference

Apply subband MVDR beamforming to an underwater acoustic 11-element ULA. Beamform the arriving signals to optimize the gain of a linear FM chirp signal arriving from 0 degrees azimuth and 0 degrees elevation. The signal has a bandwidth of 2.0 kHz. In addition, there unit amplitude 2.250 kHz interfering sine wave arriving from 28 degrees azimuth and 0 degrees elevation. Show how the MVDR beamformer nulls the interfering signal. Display the array pattern for several frequencies in the neighborhood of 2.250 kHz. The speed of sound is 1500 meters/sec.

Simulate Arriving Signal and Noise

```
array = phased.ULA('NumElements',11,'ElementSpacing',0.3);
fs = 2000;
carrierFreq = 2000;
t = (0:1/fs:2)';
sig = chirp(t,0,2,fs/2);
c = 1500;
collector = phased.WidebandCollector('Sensor',array,'PropagationSpeed',c,...
    'SampleRate',fs,'ModulatedInput',true,...
    'CarrierFrequency',carrierFreq);
incidentAngle = [0;0];
sig1 = collector(sig,incidentAngle);
noise = 0.3*(randn(size(sig1)) + 1j*randn(size(sig1)));
```

Simulate Interfering Signal

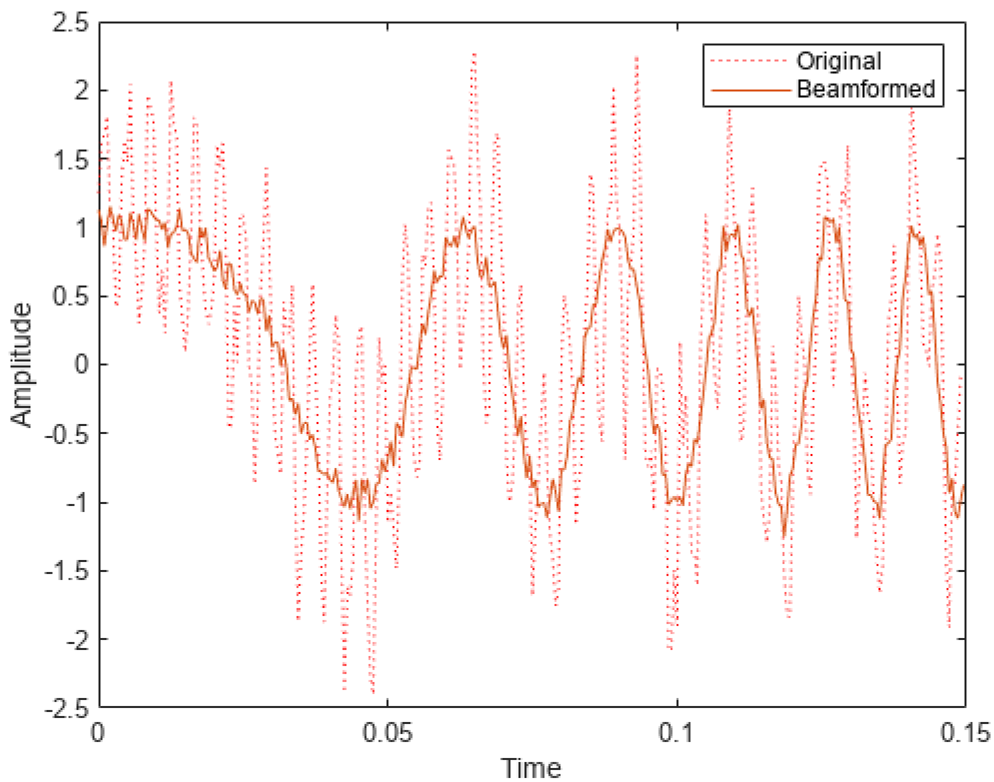
Combine both the desired and interfering signals.

```
fint = 250;
sigint = sin(2*pi*fint*t);
interfangle = [28;0];
sigint1 = collector(sigint,interfangle);
rx = sig1 + sigint1 + noise;
```

Apply MVDR Beamforming

Use the combined noise and interfering signal as training data.

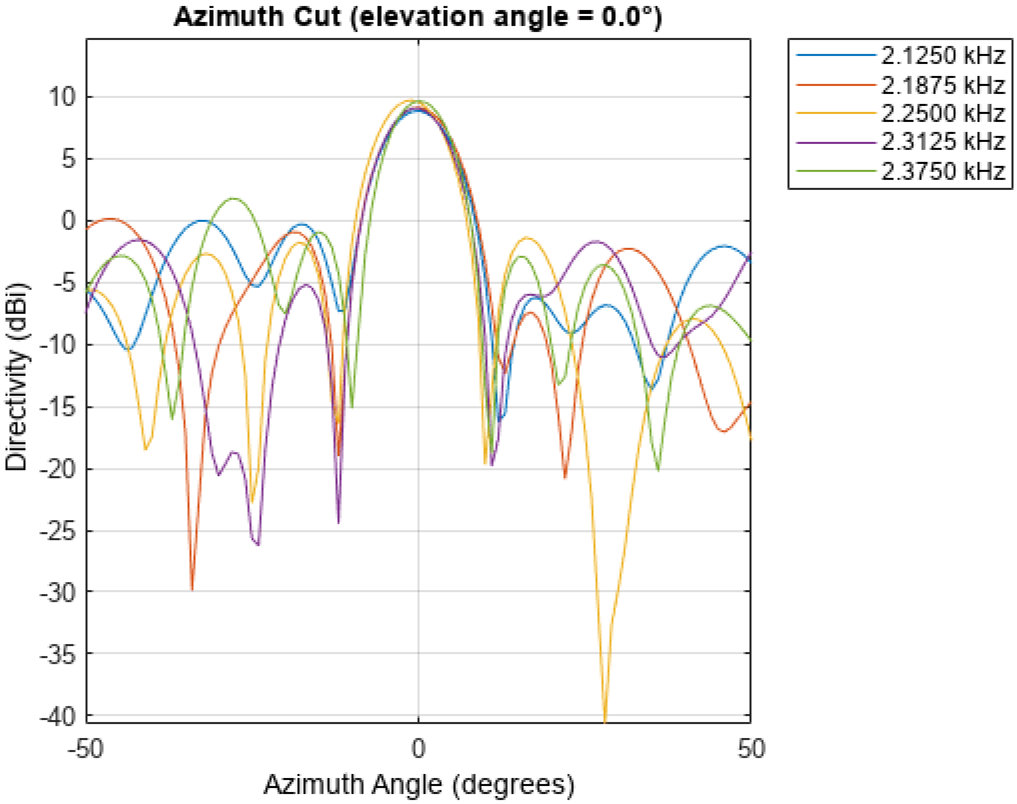
```
beamformer = phased.SubbandMVDRBeamformer('SensorArray',array,...
    'Direction',incidentAngle,'OperatingFrequency',carrierFreq,...
    'PropagationSpeed',c,'SampleRate',fs,'TrainingInputPort',true,...
    'NumSubbands',64,...
    'SubbandsOutputPort',true,'WeightsOutputPort',true);
[y,w,subbandfreq] = beamformer(rx,sigint1 + noise);
tidx = [1:300];
plot(t(tidx),real(rx(tidx,6)),'r:',t(tidx),real(y(tidx)))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Plot Array Response Showing Beampattern Null

Plot the response pattern for five bands near 2.250 kHz.

```
fdx = [5,7,9,11,13];
pattern(array,subbandfreq(fdx).',-50:50,0,...
    'PropagationSpeed',c,'Weights',w(:,fdx),...
    'CoordinateSystem','rectangular');
```



The beamformer places a null at 28 degrees for the subband containing 2.250 kHz.

Version History

Introduced in R2015b

References

[1] Proakis, J. *Digital Communications*. New York: McGraw-Hill, 2001.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill

[3] Saakian, A. *Radio Wave Propagation Fundamentals*. Norwood, MA: Artech House, 2011.

[4] Balanis, C. *Advanced Engineering Electromagnetics*. New York: Wiley & Sons, 1989.

[5] Rappaport, T. *Wireless Communications: Principles and Practice, 2nd Ed* New York: Prentice Hall, 2002.

phased.SubbandPhaseShiftBeamformer

Package: phased

Subband phase shift beamformer

Description

The SubbandPhaseShiftBeamformer object implements a subband phase shift beamformer.

To compute the beamformed signal:

1. Define and set up your subband phase shift beamformer. See "Construction" on page 1-1674.
2. Call step to perform the beamforming operation according to the properties of phased.SubbandPhaseShiftBeamformer. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

H = phased.SubbandPhaseShiftBeamformer creates a subband phase shift beamformer System object, H. The object performs subband phase shift beamforming on the received signal.

H = phased.SubbandPhaseShiftBeamformer(Name,Value) creates a subband phase shift beamformer object, H, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

SensorArray

Sensor array

Sensor array specified as an array System object belonging to the phased package. A sensor array can contain subarrays.

Default: phased.ULA with default property values

PropagationSpeed

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the beamformer in hertz as a scalar. The default value of this property corresponds to 300 MHz. This property can be specified as single or double precision.

Default: `3e8`

SampleRate

Signal sampling rate

Specify the signal sampling rate (in hertz) as a positive scalar. This property can be specified as single or double precision.

Default: `1e6`

NumSubbands

Number of subbands

Specify the number of subbands used in the subband processing as a positive integer. This property can be specified as single or double precision.

Default: `64`

DirectionSource

Source of beamforming direction

Specify whether the beamforming direction for the beamformer comes from the `Direction` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `Direction` property of this object specifies the beamforming direction.
`'Input port'`	An input argument in each invocation of `step` specifies the beamforming direction.

Default: `'Property'`

Direction

Beamforming directions

Specify the beamforming directions of the beamformer as a two-row matrix. Each column of the matrix has the form [AzimuthAngle; ElevationAngle] (in degrees). Each azimuth angle must be between –180 and 180 degrees, and each elevation angle must be between –90 and 90 degrees. This property applies when you set the `DirectionSource` property to `'Property'`. This property can be specified as single or double precision.

Default: `[0; 0]`

WeightsOutputPort

Output beamforming weights

To obtain the weights used in the beamformer, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the weights, set this property to `false`.

Default: `false`

SubbandsOutputPort

Output subband center frequencies

To obtain the center frequencies of each subband, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the center frequencies, set this property to `false`.

Default: `false`

Methods

step	Beamforming using subband phase shifting

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Subband Phase-Shift Beamformer for Underwater ULA

Apply subband phase-shift beamforming to an 11-element underwater ULA. The incident angle of a wideband signal is 10° in azimuth and 30° in elevation. The carrier frequency is 2 kHz.

Create the ULA.

```
antenna = phased.ULA('NumElements',11,'ElementSpacing',0.3);
antenna.Element.FrequencyRange = [20 20000];
```

Create a chirp signal with noise.

```
fs = 1e3;
carrierFreq = 2e3;
t = (0:1/fs:2)';
x = chirp(t,0,2,fs);
c = 1500;
collector = phased.WidebandCollector('Sensor',antenna, ...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',true,'CarrierFrequency',carrierFreq);
incidentAngle = [10;30];
x = collector(x,incidentAngle);
noise = 0.3*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;
```

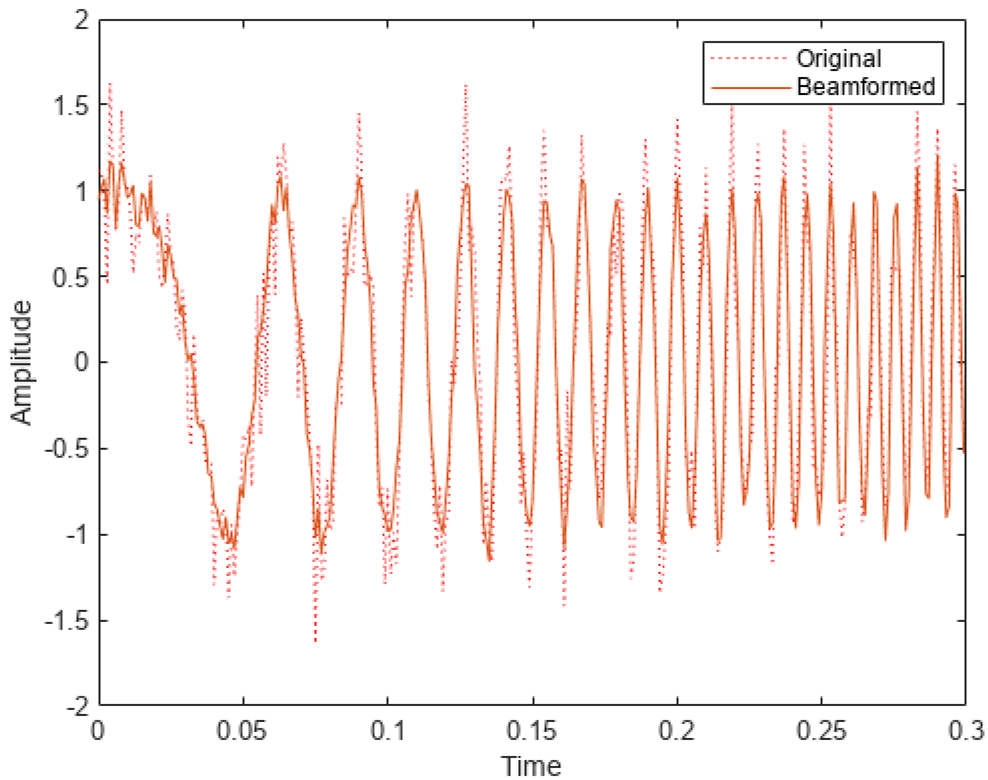
Beamform in the direction of the incident angle.

```
beamformer = phased.SubbandPhaseShiftBeamformer('SensorArray',antenna, ...
    'Direction',incidentAngle,'OperatingFrequency',carrierFreq, ...
```

```
    'PropagationSpeed',c,'SampleRate',fs,'SubbandsOutputPort',true, ...
    'WeightsOutputPort',true);
[y,w,subbandfreq] = beamformer(rx);
```

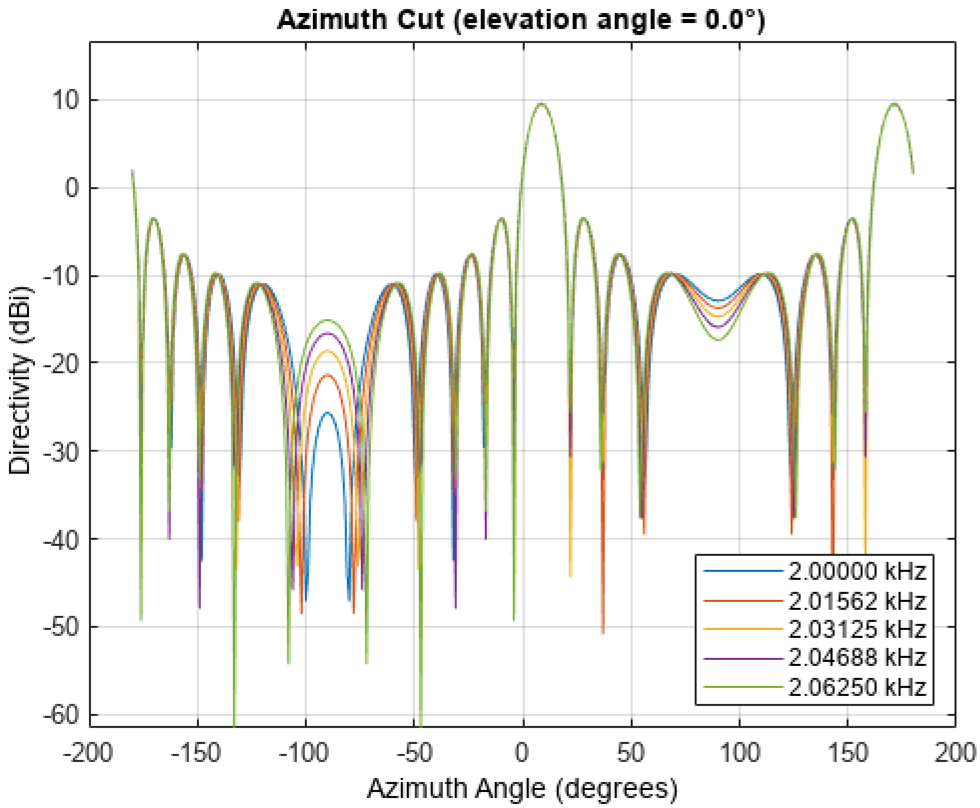
Plot the real part of the original and beamformed signals.

```
plot(t(1:300),real(rx(1:300,6)),'r:',t(1:300),real(y(1:300)))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Plot the response pattern for five frequency bands.

```
pattern(antenna,subbandfreq(1:5).',[-180:180],0,'PropagationSpeed',c, ...
    'CoordinateSystem','rectangular','Weights',w(:,1:5))
legend('location','SouthEast')
```



Algorithms

Beamforming Algorithm

The subband phase shift beamformer separates the signal into several subbands and applies narrowband phase shift beamforming to the signal in each subband. The beamformed signals in all the subbands are regrouped to form the output signal.

For further details, see [1].

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.Collector` | `phased.PhaseShiftBeamformer` | `phased.TimeDelayBeamformer` | `phased.WidebandCollector` | `uv2azel` | `phitheta2azel`

Topics

"Wideband Beamforming"

step

System object: `phased.SubbandPhaseShiftBeamformer`
Package: `phased`

Beamforming using subband phase shifting

Syntax

```
Y = step(H,X)
Y = step(H,X,ANG)
[Y,W] = step( ___ )
[Y,FREQ] = step( ___ )
[Y,W,FREQ] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` performs subband phase shift beamforming on the input, `X`, and returns the beamformed output in `Y`.

`Y = step(H,X,ANG)` uses `ANG` as the beamforming direction. This syntax is available when you set the `DirectionSource` property to `'Input port'`.

`[Y,W] = step( ___ )` returns the beamforming weights, `W`. This syntax is available when you set the `WeightsOutputPort` property to `true`.

`[Y,FREQ] = step( ___ )` returns the center frequencies of subbands, `FREQ`. This syntax is available when you set the `SubbandsOutputPort` property to `true`.

`[Y,W,FREQ] = step( ___ )` returns beamforming weights and center frequencies of subbands. This syntax is available when you set the `WeightsOutputPort` property to `true` and set the `SubbandsOutputPort` property to `true`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Beamformer object.

X

Input signal, specified as an M-by-N matrix. If the sensor array contains subarrays, N is the number of subarrays; otherwise, N is the number of elements. This argument can be specified as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

ANG

Beamforming directions, specified as a two-row matrix. Each column has the form [AzimuthAngle; ElevationAngle], in degrees. Each azimuth angle must be between –180 and 180 degrees, and each elevation angle must be between –90 and 90 degrees. This argument can be specified as single or double precision.

Output Arguments

Y

Beamformed output. `Y` is an M-by-L matrix, where M is the number of rows of `X` and L is the number of beamforming directions. This argument can be returned as single or double precision.

W

Beamforming weights. `W` has dimensions N-by-K-by-L. K is the number of subbands in the `NumSubbands` property. L is the number of beamforming directions. If the sensor array contains subarrays, N is the number of subarrays; otherwise, N is the number of elements. Each column of `W` specifies the narrowband beamforming weights used in the corresponding subband for the corresponding direction. This argument can be returned as single or double precision.

FREQ

Center frequencies of subbands. `FREQ` is a column vector of length K, where K is the number of subbands in the `NumSubbands` property. This argument can be returned as single or double precision.

Examples

Subband Phase-Shift Beamformer for Underwater ULA

Apply subband phase-shift beamforming to an 11-element underwater ULA. The incident angle of a wideband signal is 10° in azimuth and 30° in elevation. The carrier frequency is 2 kHz.

Create the ULA.

```
antenna = phased.ULA('NumElements',11,'ElementSpacing',0.3);
antenna.Element.FrequencyRange = [20 20000];
```

Create a chirp signal with noise.

```
fs = 1e3;
carrierFreq = 2e3;
```

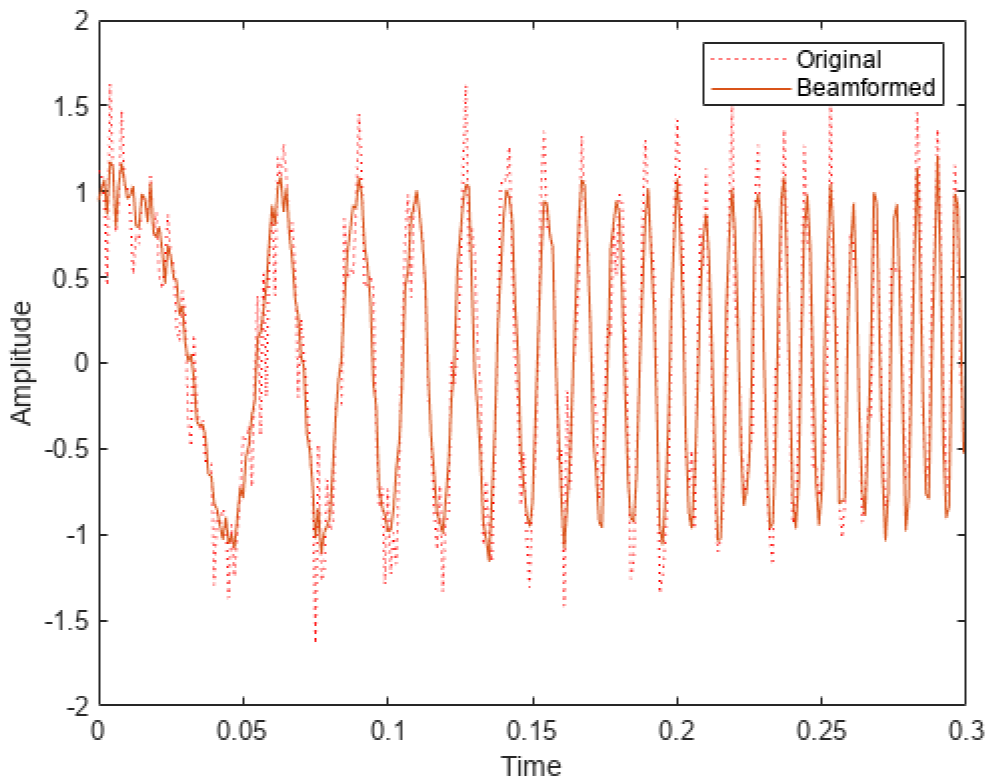
```
t = (0:1/fs:2)';
x = chirp(t,0,2,fs);
c = 1500;
collector = phased.WidebandCollector('Sensor',antenna, ...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',true,'CarrierFrequency',carrierFreq);
incidentAngle = [10;30];
x = collector(x,incidentAngle);
noise = 0.3*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;
```

Beamform in the direction of the incident angle.

```
beamformer = phased.SubbandPhaseShiftBeamformer('SensorArray',antenna, ...
    'Direction',incidentAngle,'OperatingFrequency',carrierFreq, ...
    'PropagationSpeed',c,'SampleRate',fs,'SubbandsOutputPort',true, ...
    'WeightsOutputPort',true);
[y,w,subbandfreq] = beamformer(rx);
```

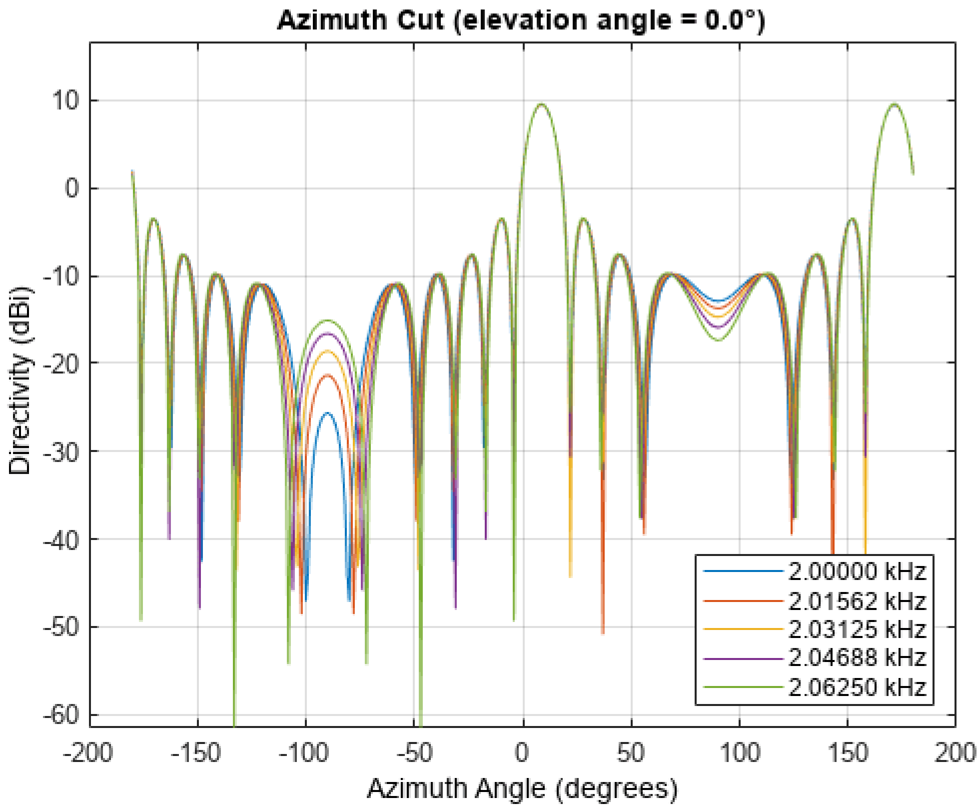
Plot the real part of the original and beamformed signals.

```
plot(t(1:300),real(rx(1:300,6)),'r:',t(1:300),real(y(1:300)))
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Plot the response pattern for five frequency bands.

```
pattern(antenna,subbandfreq(1:5).',[-180:180],0,'PropagationSpeed',c, ...
    'CoordinateSystem','rectangular','Weights',w(:,1:5))
legend('location','SouthEast')
```



Algorithms

The subband phase shift beamformer separates the signal into several subbands and applies narrowband phase shift beamforming to the signal in each subband. The beamformed signals in all the subbands are regrouped to form the output signal.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

phased.SumDifferenceMonopulseTracker

Package: `phased`

Sum and difference monopulse for ULA

Description

The `SumDifferenceMonopulseTracker` object implements a sum and difference monopulse algorithm on a uniform linear array.

To estimate the direction of arrival (DOA):

1. Define and set up your sum and difference monopulse DOA estimator. See "Construction" on page 1-1684.
2. Call `step` to estimate the DOA according to the properties of `phased.SumDifferenceMonopulseTracker`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.SumDifferenceMonopulseTracker` creates a tracker System object, `H`. The object uses sum and difference monopulse algorithms on a uniform linear array (ULA).

`H = phased.SumDifferenceMonopulseTracker(Name,Value)` creates a ULA monopulse tracker object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be a `phased.ULA` object.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

`OperatingFrequency`

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

`NumPhaseShifterBits`

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed. You can specify this property as single or double precision.

Default: `0`

Methods

step	Perform monopulse tracking using ULA

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Find Target Direction Using Monopulse Tracker

Determine the direction of a target at a 60.1° broadside angle to a ULA starting with an approximate direction of 60°

```
array = phased.ULA('NumElements',4);
steervec = phased.SteeringVector('SensorArray',array);
tracker = phased.SumDifferenceMonopulseTracker('SensorArray',array);
x = steervec(tracker.OperatingFrequency,60.1).';
est_dir = tracker(x,60)
```

```
est_dir = 60.1000
```

Algorithms

Monopulse Algorithm

The sum-and-difference monopulse algorithm is used to the estimate the arrival direction of a narrowband signal impinging upon a uniform linear array (ULA). First, compute the conventional response of an array steered to an arrival direction φ_0. For a ULA, the arrival direction is specified by the broadside angle. To specify that the maximum response axis (MRA) point towards the φ_0 direction, set the weights to be

$$\mathbf{w}_s = \left(1, e^{ikd\sin\phi_0}, e^{ik2d\sin\phi_0}, \ldots, e^{ik(N-1)d\sin\phi_0}\right)$$

where *d* is the element spacing and $k = 2\pi/\lambda$ is the wavenumber. An incoming plane wave, coming from any arbitrary direction φ, is represented by

$$\mathbf{v} = \left(1, e^{ikd\sin\phi}, e^{ik2d\sin\phi}, \ldots, e^{ik(N-1)d\sin\phi}\right)$$

The conventional response of this array to any incoming plane wave is given by $\mathbf{w}_s^H\mathbf{v}(\varphi)$ and is shown in the polar plot below as the *Sum Pattern*. The array is designed to steer towards $\varphi_0 = 30°$.

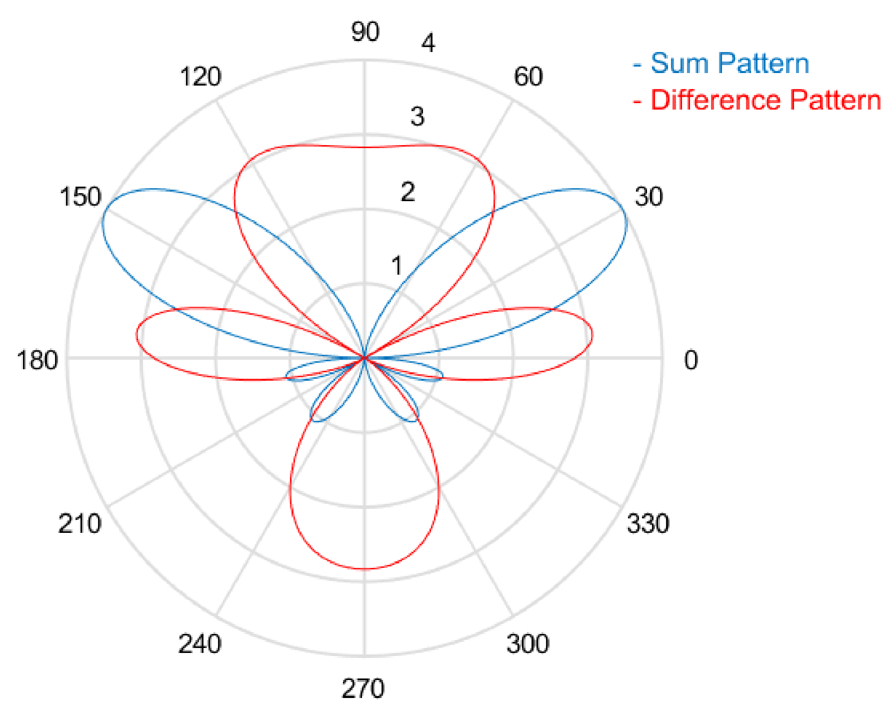
The second pattern, called the *Difference Pattern*, is obtained by using phased-reversed weights. The weights are determined by phase-reversing the latter half of the conventional steering vector. For an array with an even number of elements, the phase-reversed weights are

$$\mathbf{w}_d = -i\left(1, e^{ikd\sin\phi_0}, e^{ik2d\sin\phi_0}, \ldots, e^{ikN/2d\sin\phi_0}, -e^{ik(N/2+1)d\sin\phi_0}, \ldots, -e^{ik(N-1)d\sin\phi_0}\right)$$

(For an array with an odd number of elements, the middle weight is set to zero). The multiplicative factor –*i* is used for convenience. The response of the difference array to the incoming vector is

$$\mathbf{w}_d^H\mathbf{v}(\varphi)$$

This figure shows the sum and difference beam patterns of a four-element uniform linear array (ULA) steered 30° from broadside. The array elements are spaced at one-half wavelength. The sum pattern shows that the array has its maximum response at 30° and the difference pattern has a null at 30°.



The monopulse response curve is obtained by dividing the difference pattern by the sum pattern and taking the real part.

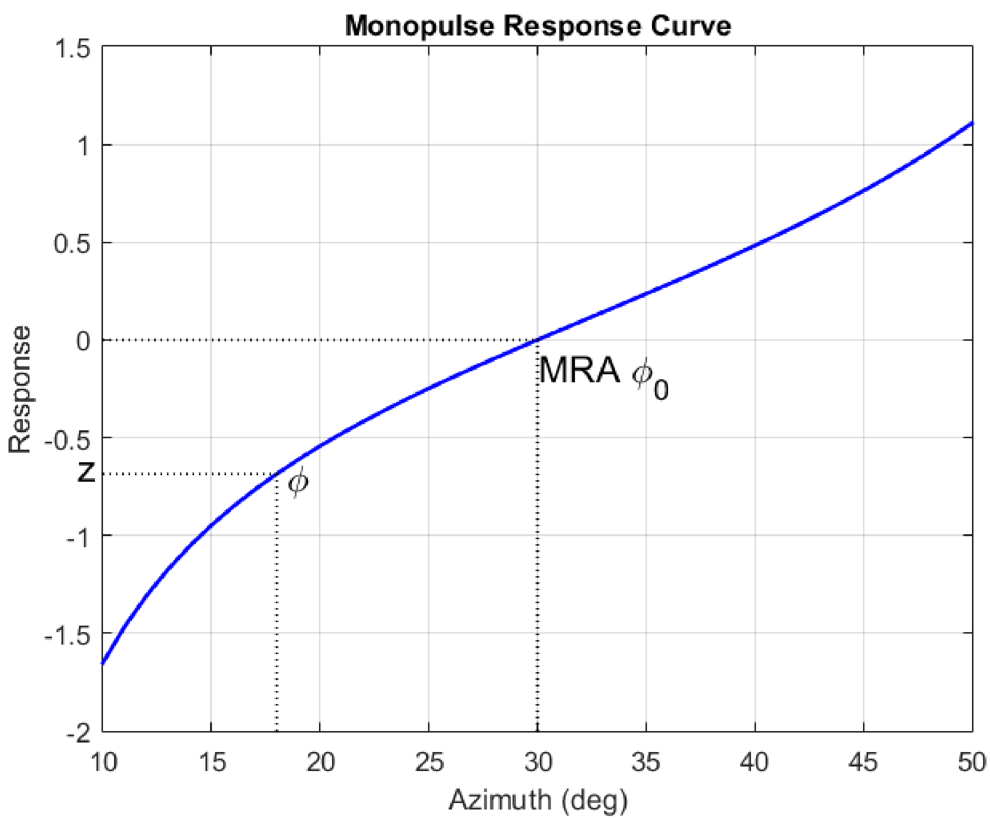
$$R(\varphi) = Re\left(\frac{\mathbf{w}_d^H \mathbf{v}(\varphi)}{\mathbf{w}_s^H \mathbf{v}(\varphi)}\right)$$

To use the monopulse response curve to obtain the arrival angle, φ, of a narrowband signal, $\mathbf{x}$, compute

$$z = Re\left(\frac{\mathbf{w}_d^H \mathbf{x}}{\mathbf{w}_s^H \mathbf{x}}\right)$$

and invert the response curve, $\varphi = R^{-1}(z)$, to obtain φ.

The response curve is not generally single valued and can only be inverted when arrival angles lie within the main lobe where it is single valued This figure shows the monopulse response curve within the main lobe of the four-element ULA array.



There are two desirable properties of the monopulse response curve. The first is that it have a steep slope. A steep slope insures robustness against noise. The second property is that the mainlobe be as wide as possible. A steep slope is ensure by a larger array but leads to a smaller mainlobe. You will need to trade off one property with the other.

For further details, see [1].

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Seliktar, Y. *Space-Time Adaptive Monopulse Processing*. Ph.D. Thesis. Georgia Institute of Technology, Atlanta, 1998.

[2] Rhodes, D. *Introduction to Monopulse*. Dedham, MA: Artech House, 1980.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.BeamscanEstimator` | `phased.SumDifferenceMonopulseTracker2D`

Topics

“Target Tracking Using Sum-Difference Monopulse Radar”

step

System object: `phased.SumDifferenceMonopulseTracker`
Package: `phased`

Perform monopulse tracking using ULA

Syntax

`ESTANG = step(H,X,STANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`ESTANG = step(H,X,STANG)` estimates the incoming direction `ESTANG` of the input signal, `X`, based on an initial guess of the direction.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Tracker object of type `phased.SumDifferenceMonopulseTracker`.

X

Input signal, specified as a row vector whose number of columns corresponds to number of channels. You can specify this argument as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

STANG

Initial guess of the direction, specified as a scalar that represents the broadside angle in degrees. A typical initial guess is the current steering angle. The value of `STANG` is between –90 and 90. The angle is defined in the array's local coordinate system. For details regarding the local coordinate system of the ULA, type `phased.ULA.coordinateSystemInfo`. You can specify this argument as single or double precision.

Output Arguments

ESTANG

Estimate of incoming direction, returned as a scalar that represents the broadside angle in degrees. The value is between –90 and 90. The angle is defined in the array's local coordinate system.

Examples

Find Target Direction Using Monopulse Tracker

Determine the direction of a target at a 60.1° broadside angle to a ULA starting with an approximate direction of 60°

```
array = phased.ULA('NumElements',4);
steervec = phased.SteeringVector('SensorArray',array);
tracker = phased.SumDifferenceMonopulseTracker('SensorArray',array);
x = steervec(tracker.OperatingFrequency,60.1).';
est_dir = tracker(x,60)
```

```
est_dir = 60.1000
```

Algorithms

The sum-and-difference monopulse algorithm is used to the estimate the arrival direction of a narrowband signal impinging upon a uniform linear array (ULA). First, compute the conventional response of an array steered to an arrival direction φ_0. For a ULA, the arrival direction is specified by the broadside angle. To specify that the maximum response axis (MRA) point towards the φ_0 direction, set the weights to be

$$\mathbf{w}_s = \left(1, e^{ikd\sin\phi_0}, e^{ik2d\sin\phi_0}, \ldots, e^{ik(N-1)d\sin\phi_0}\right)$$

where d is the element spacing and $k = 2\pi/\lambda$ is the wavenumber. An incoming plane wave, coming from any arbitrary direction φ, is represented by

$$\mathbf{v} = \left(1, e^{ikd\sin\phi}, e^{ik2d\sin\phi}, \ldots, e^{ik(N-1)d\sin\phi}\right)$$

The conventional response of this array to any incoming plane wave is given by $\mathbf{w}_s^H\mathbf{v}(\varphi)$ and is shown in the polar plot below as the *Sum Pattern*. The array is designed to steer towards $\varphi_0 = 30°$.

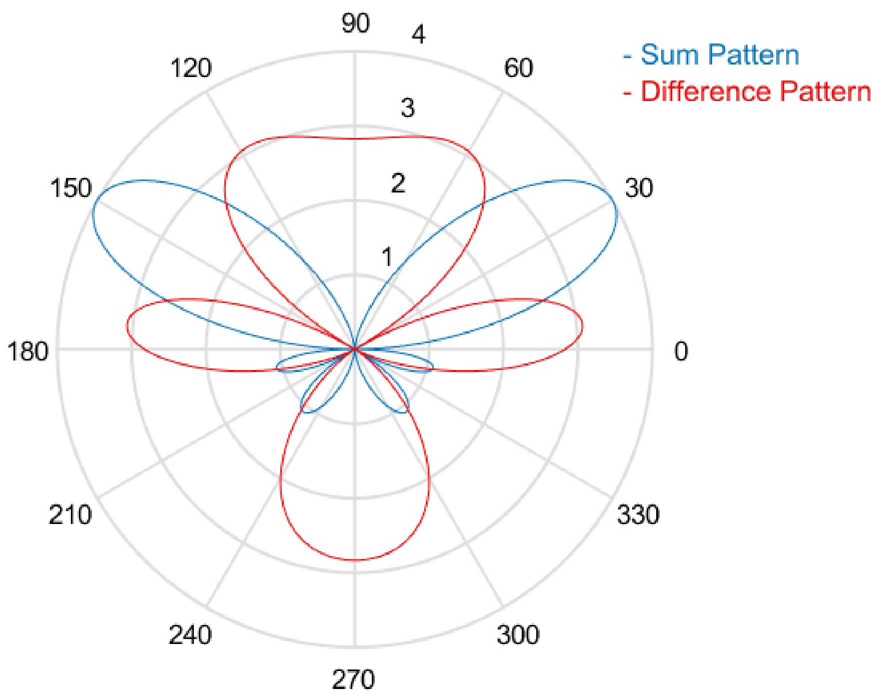
The second pattern, called the *Difference Pattern*, is obtained by using phased-reversed weights. The weights are determined by phase-reversing the latter half of the conventional steering vector. For an array with an even number of elements, the phase-reversed weights are

$$\mathbf{w}_d = -i\left(1, e^{ikd\sin\phi_0}, e^{ik2d\sin\phi_0}, \ldots, e^{ikN/2d\sin\phi_0}, -e^{ik(N/2+1)d\sin\phi_0}, \ldots, -e^{ik(N-1)d\sin\phi_0}\right)$$

(For an array with an odd number of elements, the middle weight is set to zero). The multiplicative factor –i is used for convenience. The response of the difference array to the incoming vector is

$$\mathbf{w}_d^H\mathbf{v}(\varphi)$$

This figure shows the sum and difference beam patterns of a four-element uniform linear array (ULA) steered 30° from broadside. The array elements are spaced at one-half wavelength. The sum pattern shows that the array has its maximum response at 30° and the difference pattern has a null at 30°.



The monopulse response curve is obtained by dividing the difference pattern by the sum pattern and taking the real part.

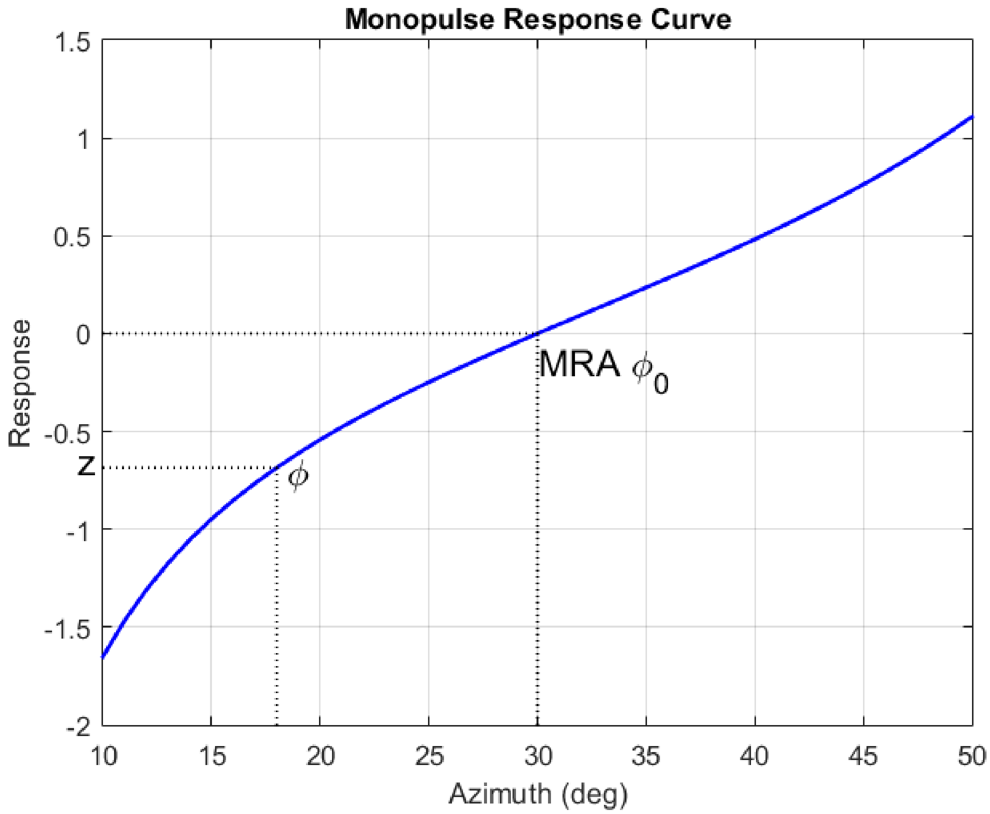
$$R(\varphi) = Re\left(\frac{\mathbf{w}_d^H \mathbf{v}(\varphi)}{\mathbf{w}_s^H \mathbf{v}(\varphi)}\right)$$

To use the monopulse response curve to obtain the arrival angle, φ, of a narrowband signal, $\mathbf{x}$, compute

$$z = Re\left(\frac{\mathbf{w}_d^H \mathbf{x}}{\mathbf{w}_s^H \mathbf{x}}\right)$$

and invert the response curve, $\varphi = R^{-1}(z)$, to obtain φ.

The response curve is not generally single valued and can only be inverted when arrival angles lie within the main lobe where it is single valued This figure shows the monopulse response curve within the main lobe of the four-element ULA array.



There are two desirable properties of the monopulse response curve. The first is that it have a steep slope. A steep slope insures robustness against noise. The second property is that the mainlobe be as wide as possible. A steep slope is ensure by a larger array but leads to a smaller mainlobe. You will need to trade off one property with the other.

For further details, see [1].

References

[1] Seliktar, Y. *Space-Time Adaptive Monopulse Processing*. Ph.D. Thesis. Georgia Institute of Technology, Atlanta, 1998.

[2] Rhodes, D. *Introduction to Monopulse*. Dedham, MA: Artech House, 1980.

phased.SumDifferenceMonopulseTracker2D

Package: `phased`

Sum and difference monopulse for URA

Description

The `SumDifferenceMonopulseTracker2D` object implements a sum and difference monopulse algorithm for a uniform rectangular array.

To estimate the direction of arrival (DOA):

1. Define and set up your sum and difference monopulse DOA estimator. See "Construction" on page 1-1693.
2. Call `step` to estimate the DOA according to the properties of `phased.SumDifferenceMonopulseTracker2D`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.SumDifferenceMonopulseTracker2D` creates a tracker System object, `H`. The object uses sum and difference monopulse algorithms on a uniform rectangular array (URA).

`H = phased.SumDifferenceMonopulseTracker2D(Name,Value)` creates a URA monopulse tracker object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be a `phased.URA` object.

Default: `phased.URA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

OperatingFrequency

System operating frequency

Specify the operating frequency of the system in hertz as a positive scalar. The default value corresponds to 300 MHz. You can specify this property as single or double precision.

Default: `3e8`

NumPhaseShifterBits

Number of phase shifter quantization bits

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed. You can specify this property as single or double precision.

Default: `0`

Methods

step	Perform monopulse tracking using URA

Common to All System Objects	
release	Allow System object property value changes

Examples

Find Target Direction Using Sum-Difference 2D Monopulse Tracker

Using a URA, determine the direction of a target at approximately 60° azimuth and 20° elevation.

```
array = phased.URA('Size',4);
steeringvec = phased.SteeringVector('SensorArray',array);
tracker = phased.SumDifferenceMonopulseTracker2D('SensorArray',array);
x = steeringvec(tracker.OperatingFrequency,[60.1; 19.5]).';
est_dir = tracker(x,[60; 20])
```

```
est_dir = 2×1

   60.1000
   19.5000
```

Algorithms

Monopulse Algorithm

The sum-and-difference monopulse algorithm is used to the estimate the arrival direction of a narrowband signal impinging upon a uniform linear array (ULA). First, compute the conventional response of an array steered to an arrival direction φ_0. For a ULA, the arrival direction is specified by the broadside angle. To specify that the maximum response axis (MRA) point towards the φ_0 direction, set the weights to be

$$\mathbf{w}_s = \left(1, e^{ikd\sin\phi_0}, e^{ik2d\sin\phi_0}, \ldots, e^{ik(N-1)d\sin\phi_0}\right)$$

where d is the element spacing and $k = 2\pi/\lambda$ is the wavenumber. An incoming plane wave, coming from any arbitrary direction φ, is represented by

$$\mathbf{v} = \left(1, e^{ikd\sin\phi}, e^{ik2d\sin\phi}, \ldots, e^{ik(N-1)d\sin\phi}\right)$$

The conventional response of this array to any incoming plane wave is given by $\mathbf{w}_s^H\mathbf{v}(\varphi)$ and is shown in the polar plot below as the *Sum Pattern*. The array is designed to steer towards $\varphi_0 = 30°$.

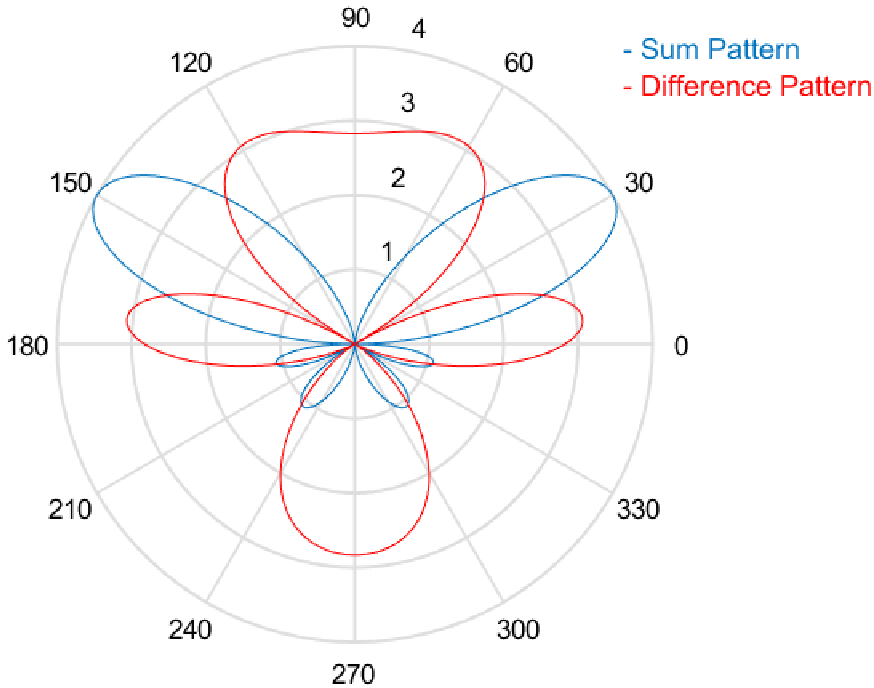
The second pattern, called the *Difference Pattern*, is obtained by using phased-reversed weights. The weights are determined by phase-reversing the latter half of the conventional steering vector. For an array with an even number of elements, the phase-reversed weights are

$$\mathbf{w}_d = -i\left(1, e^{ikd\sin\phi_0}, e^{ik2d\sin\phi_0}, \ldots, e^{ikN/2d\sin\phi_0}, -e^{ik(N/2+1)d\sin\phi_0}, \ldots, -e^{ik(N-1)d\sin\phi_0}\right)$$

(For an array with an odd number of elements, the middle weight is set to zero). The multiplicative factor $-i$ is used for convenience. The response of the difference array to the incoming vector is

$$\mathbf{w}_d^H\mathbf{v}(\varphi)$$

This figure shows the sum and difference beam patterns of a four-element uniform linear array (ULA) steered 30° from broadside. The array elements are spaced at one-half wavelength. The sum pattern shows that the array has its maximum response at 30° and the difference pattern has a null at 30°.



The monopulse response curve is obtained by dividing the difference pattern by the sum pattern and taking the real part.

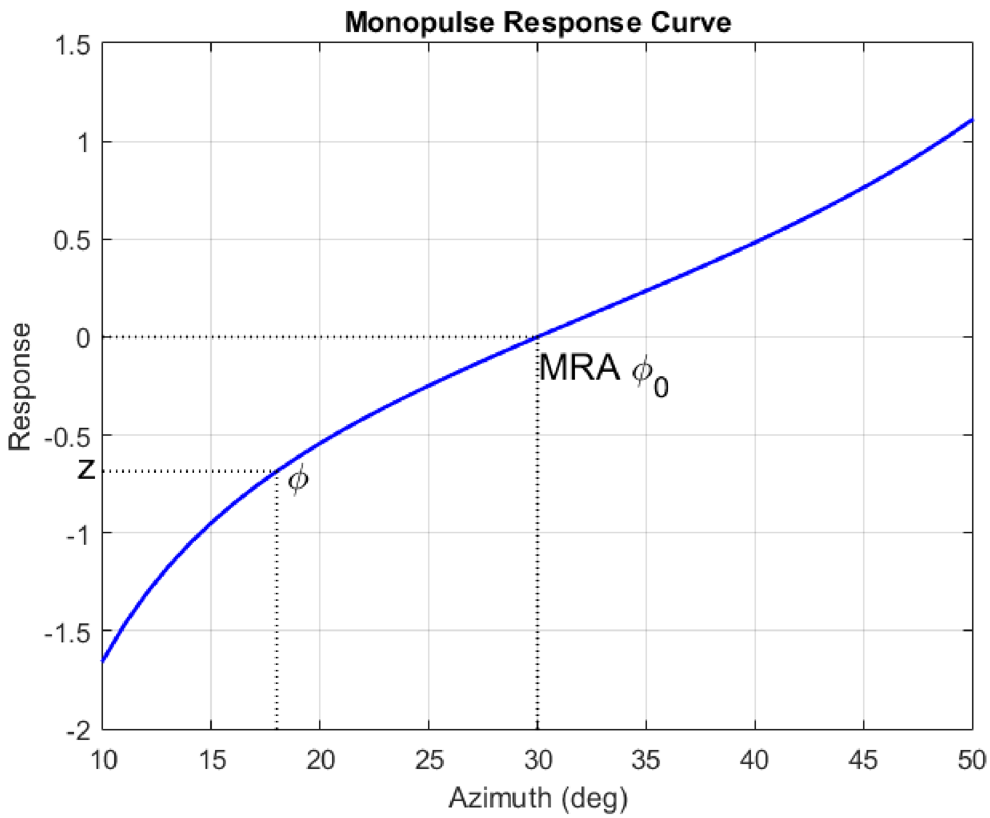
$$R(\varphi) = Re\left(\frac{\mathbf{w}_d^H \mathbf{v}(\varphi)}{\mathbf{w}_s^H \mathbf{v}(\varphi)}\right)$$

To use the monopulse response curve to obtain the arrival angle, φ, of a narrowband signal, $\mathbf{x}$, compute

$$z = Re\left(\frac{\mathbf{w}_d^H \mathbf{x}}{\mathbf{w}_s^H \mathbf{x}}\right)$$

and invert the response curve, $\varphi = R^{-1}(z)$, to obtain φ.

The response curve is not generally single valued and can only be inverted when arrival angles lie within the main lobe where it is single valued This figure shows the monopulse response curve within the main lobe of the four-element ULA array.



There are two desirable properties of the monopulse response curve. The first is that it have a steep slope. A steep slope insures robustness against noise. The second property is that the mainlobe be as wide as possible. A steep slope is ensure by a larger array but leads to a smaller mainlobe. You will need to trade off one property with the other.

For further details, see [1].

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Seliktar, Y. *Space-Time Adaptive Monopulse Processing*. Ph.D. Thesis. Georgia Institute of Technology, Atlanta, 1998.

[2] Rhodes, D. *Introduction to Monopulse*. Dedham, MA: Artech House, 1980.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also
`phased.BeamscanEstimator` | `phased.SumDifferenceMonopulseTracker`

step

System object: `phased.SumDifferenceMonopulseTracker2D`
Package: `phased`

Perform monopulse tracking using URA

Syntax

`ESTANG = step(H,X,STANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`ESTANG = step(H,X,STANG)` estimates the incoming direction `ESTANG` of the input signal, `X`, based on an initial guess of the direction.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Tracker object of type `phased.SumDifferenceMonopulseTracker2D`.

X

Input signal, specified as a row vector whose number of columns corresponds to number of channels. You can specify this argument as single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

STANG

Initial guess of the direction, specified as a 2-by-1 vector in the form `[AzimuthAngle; ElevationAngle]` in degrees. A typical initial guess is the current steering angle. Azimuth angles must be between –180 and 180. Elevation angles must be between –90 and 90. Angles are measured in the local coordinate system of the array. For details regarding the local coordinate system of the

URA, type `phased.URA.coordinateSystemInfo`. You can specify this argument as single or double precision.

Output Arguments

ESTANG

Estimate of incoming direction, returned as a 2-by-1 vector in the form `[AzimuthAngle; ElevationAngle]` in degrees. Azimuth angles are between –180 and 180. Elevation angles are between –90 and 90. Angles are measured in the local coordinate system of the array.

Examples

Find Target Direction Using Sum-Difference 2D Monopulse Tracker

Using a URA, determine the direction of a target at approximately 60° azimuth and 20° elevation.

```
array = phased.URA('Size',4);
steeringvec = phased.SteeringVector('SensorArray',array);
tracker = phased.SumDifferenceMonopulseTracker2D('SensorArray',array);
x = steeringvec(tracker.OperatingFrequency,[60.1; 19.5]).';
est_dir = tracker(x,[60; 20])
```

```
est_dir = 2×1

   60.1000
   19.5000
```

Algorithms

The sum-and-difference monopulse algorithm is used to the estimate the arrival direction of a narrowband signal impinging upon a uniform linear array (ULA). First, compute the conventional response of an array steered to an arrival direction φ_0. For a ULA, the arrival direction is specified by the broadside angle. To specify that the maximum response axis (MRA) point towards the φ_0 direction, set the weights to be

$$\mathbf{w}_s = \left(1, e^{ikd\sin\phi_0}, e^{ik2d\sin\phi_0}, \ldots, e^{ik(N-1)d\sin\phi_0}\right)$$

where d is the element spacing and $k = 2\pi/\lambda$ is the wavenumber. An incoming plane wave, coming from any arbitrary direction φ, is represented by

$$\mathbf{v} = \left(1, e^{ikd\sin\phi}, e^{ik2d\sin\phi}, \ldots, e^{ik(N-1)d\sin\phi}\right)$$

The conventional response of this array to any incoming plane wave is given by $\mathbf{w}_s^H\mathbf{v}(\varphi)$ and is shown in the polar plot below as the *Sum Pattern*. The array is designed to steer towards $\varphi_0 = 30°$.

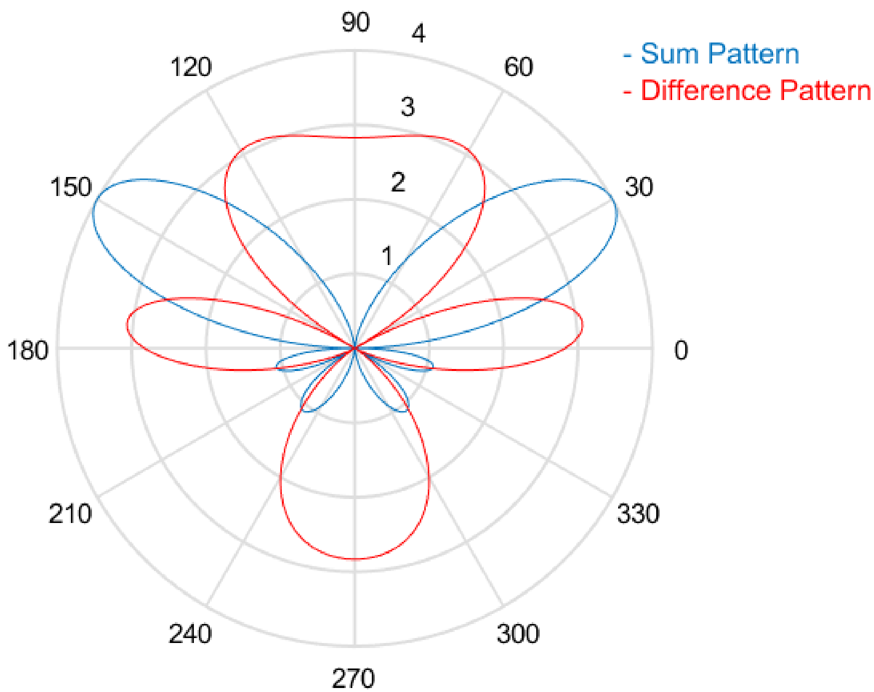
The second pattern, called the *Difference Pattern*, is obtained by using phased-reversed weights. The weights are determined by phase-reversing the latter half of the conventional steering vector. For an array with an even number of elements, the phase-reversed weights are

$$\mathbf{w}_d = -i\left(1, e^{ikd\sin\phi_0}, e^{ik2d\sin\phi_0}, \ldots, e^{ikN/2d\sin\phi_0}, -e^{ik(N/2+1)d\sin\phi_0}, \ldots, -e^{ik(N-1)d\sin\phi_0}\right)$$

(For an array with an odd number of elements, the middle weight is set to zero). The multiplicative factor –*i* is used for convenience. The response of the difference array to the incoming vector is

$$\mathbf{w}_d^H \mathbf{v}(\varphi)$$

This figure shows the sum and difference beam patterns of a four-element uniform linear array (ULA) steered 30° from broadside. The array elements are spaced at one-half wavelength. The sum pattern shows that the array has its maximum response at 30° and the difference pattern has a null at 30°.



The monopulse response curve is obtained by dividing the difference pattern by the sum pattern and taking the real part.

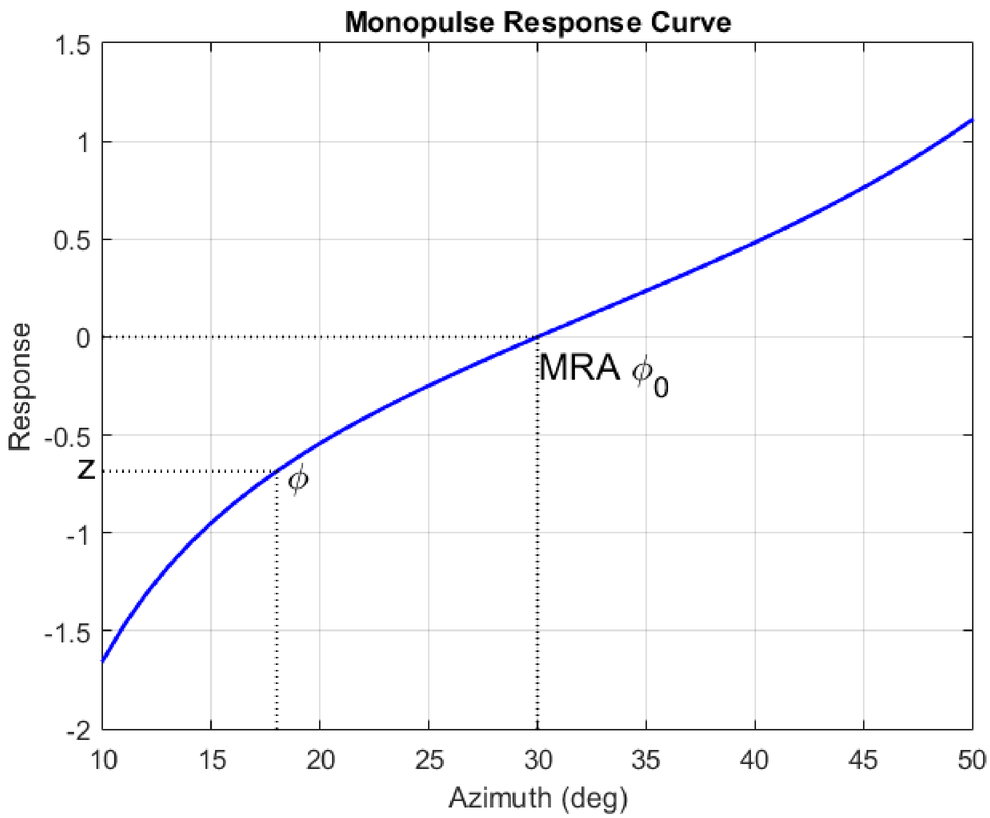
$$R(\varphi) = Re\left(\frac{\mathbf{w}_d^H \mathbf{v}(\varphi)}{\mathbf{w}_s^H \mathbf{v}(\varphi)}\right)$$

To use the monopulse response curve to obtain the arrival angle, φ, of a narrowband signal, $\mathbf{x}$, compute

$$z = Re\left(\frac{\mathbf{w}_d^H \mathbf{x}}{\mathbf{w}_s^H \mathbf{x}}\right)$$

and invert the response curve, $\varphi = R^{-1}(z)$, to obtain φ.

The response curve is not generally single valued and can only be inverted when arrival angles lie within the main lobe where it is single valued This figure shows the monopulse response curve within the main lobe of the four-element ULA array.



There are two desirable properties of the monopulse response curve. The first is that it have a steep slope. A steep slope insures robustness against noise. The second property is that the mainlobe be as wide as possible. A steep slope is ensure by a larger array but leads to a smaller mainlobe. You will need to trade off one property with the other.

For further details, see [1].

References

[1] Seliktar, Y. *Space-Time Adaptive Monopulse Processing*. Ph.D. Thesis. Georgia Institute of Technology, Atlanta, 1998.

[2] Rhodes, D. *Introduction to Monopulse*. Dedham, MA: Artech House, 1980.

See Also

`uv2azel` | `phitheta2azel` | `azel2uv` | `azel2phitheta`

phased.TimeDelayBeamformer

Package: `phased`

Time delay beamformer

Description

The `TimeDelayBeamformer` object implements a time delay beamformer.

To compute the beamformed signal:

1. Define and set up your time delay beamformer. See "Construction" on page 1-1703.
2. Call `step` to perform the beamforming operation according to the properties of `phased.TimeDelayBeamformer`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.TimeDelayBeamformer` creates a time delay beamformer System object, `H`. The object performs delay and sum beamforming on the received signal using time delays.

`H = phased.TimeDelayBeamformer(Name,Value)` creates a time delay beamformer object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be an array object in the `phased` package. The array cannot contain subarrays.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

SampleRate

Signal sampling rate

Specify the signal sampling rate (in hertz) as a positive scalar. This property can be specified as single or double precision.

Default: `1e6`

DirectionSource

Source of beamforming direction

Specify whether the beamforming direction comes from the `Direction` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `Direction` property of this object specifies the beamforming direction.
`'Input port'`	An input argument in each invocation of `step` specifies the beamforming direction.

Default: `'Property'`

Direction

Beamforming direction

Specify the beamforming direction of the beamformer as a column vector of length 2. The direction is specified in the format of `[AzimuthAngle; ElevationAngle]` (in degrees). The azimuth angle is between –180 and 180. The elevation angle is between –90 and 90. This property applies when you set the `DirectionSource` property to `'Property'`. This property can be specified as single or double precision.

Default: `[0; 0]`

WeightsOutputPort

Output beamforming weights

To obtain the weights used in the beamformer, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the weights, set this property to `false`.

Default: `false`

Methods

step	Perform time delay beamforming

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Time-Delay Beamformer Applied to ULA

Apply a time-delay beamformer to an 11-element uniform linear acoustic array. The arrival angle of the signal is -50 degrees in azimuth and 30 degrees in elevation. The arriving signal is a 0.3 second segment of a linear FM chirp having a 500 Hz bandwidth. Assume the speed of sound in air is 340.0 m/s.

Simulate the arriving signal at the wideband collector.

```
microphone = phased.CustomMicrophoneElement('FrequencyVector',[20,20000],'FrequencyResponse',[1,1
array = phased.ULA('Element',microphone,'NumElements',11,'ElementSpacing',0.04);
fs = 8000;
t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340;
collector = phased.WidebandCollector('Sensor',array,...
    'PropagationSpeed',c,'SampleRate',fs,'ModulatedInput',false);
incidentAngle = [-50;30];
x = collector(x.',incidentAngle);
```

Add white Gaussian random noise to the signal.

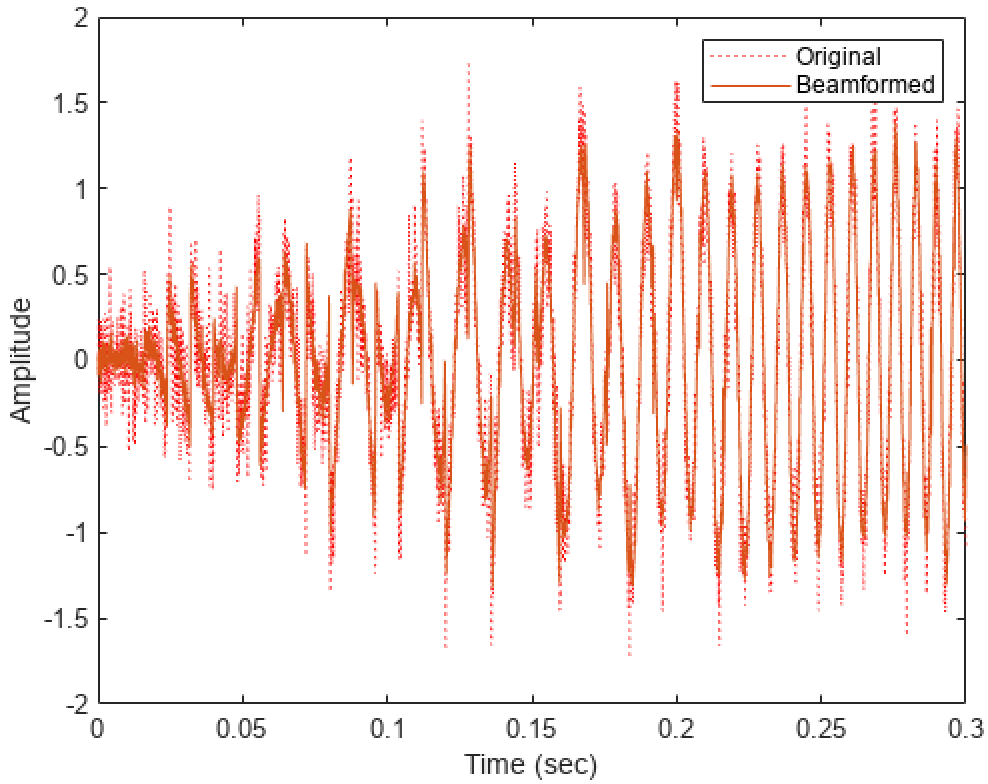
```
sigma = 0.2;
noise = sigma*randn(size(x));
rx = x + noise;
```

Beamform the incident signals using a time-delay beamformer.

```
beamformer = phased.TimeDelayBeamformer('SensorArray',array,...
    'SampleRate',fs,'PropagationSpeed',c,...
    'Direction',incidentAngle);
y = beamformer(rx);
```

Plot the beamformed signal against the incident signal at the middle sensor of the array.

```
plot(t,rx(:,6),'r:',t,y)
xlabel('Time (sec)')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Requires dynamic memory allocation. See “Limitations for System Objects that Require Dynamic Memory Allocation”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.FrostBeamformer` | `phased.PhaseShiftBeamformer` | `phased.SubbandPhaseShiftBeamformer` | `phased.TimeDelayLCMVBeamformer` | `uv2azel` | `phitheta2azel`

Topics

“Wideband Beamforming”

step

System object: `phased.TimeDelayBeamformer`
Package: `phased`

Perform time delay beamforming

Syntax

```
Y = step(H,X)
Y = step(H,X,ANG)
[Y,W] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` performs time delay beamforming on the input, `X`, and returns the beamformed output in `Y`. `X` is an M-by-N matrix where N is the number of elements of the sensor array. `Y` is a column vector of length M.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`Y = step(H,X,ANG)` uses `ANG` as the beamforming direction. This syntax is available when you set the `DirectionSource` property to `'Input port'`. `ANG` is a column vector of length 2 in the form of `[AzimuthAngle; ElevationAngle]` (in degrees). The azimuth angle must be between –180 and 180 degrees, and the elevation angle must be between –90 and 90 degrees.

`[Y,W] = step( ___ )` returns additional output, `W`, as the beamforming weights. This syntax is available when you set the `WeightsOutputPort` property to `true`. `W` is a column vector of length N. For a time delay beamformer, the weights are constant because the beamformer simply adds all the channels together and scales the result to preserve the signal power.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

All input and output arguments can be single or double precision.

Examples

Time-Delay Beamformer Applied to ULA

Apply a time-delay beamformer to an 11-element uniform linear acoustic array. The arrival angle of the signal is -50 degrees in azimuth and 30 degrees in elevation. The arriving signal is a 0.3 second segment of a linear FM chirp having a 500 Hz bandwidth. Assume the speed of sound in air is 340.0 m/s.

Simulate the arriving signal at the wideband collector.

```
microphone = phased.CustomMicrophoneElement('FrequencyVector',[20,20000],'FrequencyResponse',[1,
array = phased.ULA('Element',microphone,'NumElements',11,'ElementSpacing',0.04);
fs = 8000;
t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340;
collector = phased.WidebandCollector('Sensor',array,...
    'PropagationSpeed',c,'SampleRate',fs,'ModulatedInput',false);
incidentAngle = [-50;30];
x = collector(x.',incidentAngle);
```

Add white Gaussian random noise to the signal.

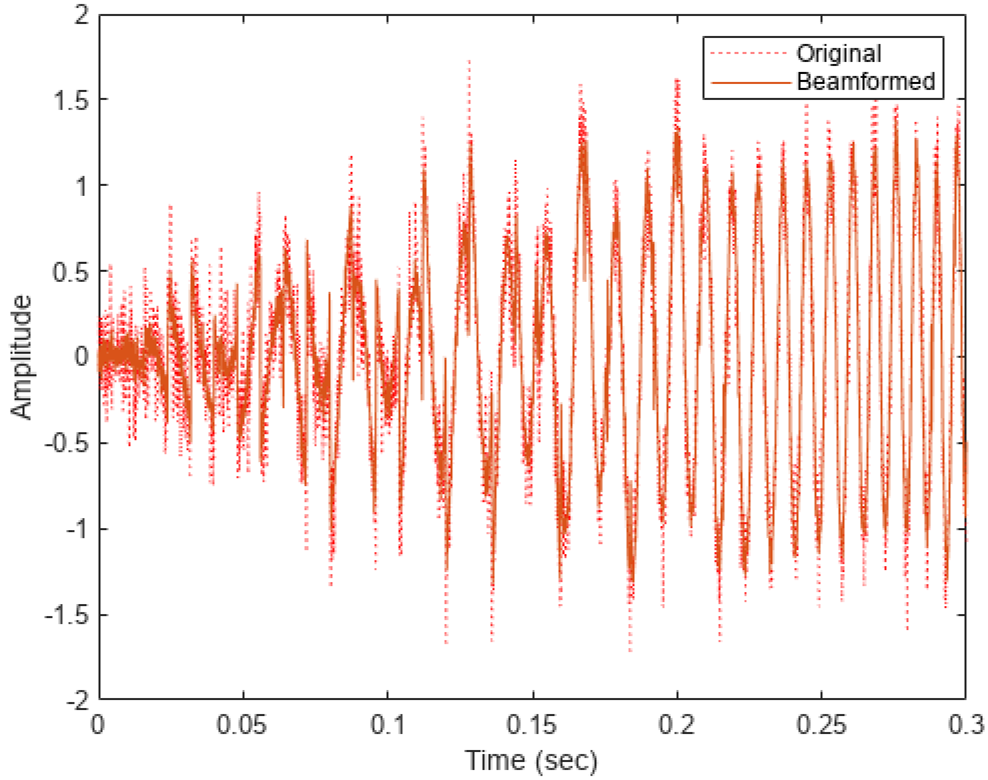
```
sigma = 0.2;
noise = sigma*randn(size(x));
rx = x + noise;
```

Beamform the incident signals using a time-delay beamformer.

```
beamformer = phased.TimeDelayBeamformer('SensorArray',array,...
    'SampleRate',fs,'PropagationSpeed',c,...
    'Direction',incidentAngle);
y = beamformer(rx);
```

Plot the beamformed signal against the incident signal at the middle sensor of the array.

```
plot(t,rx(:,6),'r:',t,y)
xlabel('Time (sec)')
ylabel('Amplitude')
legend('Original','Beamformed')
```



See Also

`uv2azel` | `phitheta2azel`

phased.TimeDelayLCMVBeamformer

Package: `phased`

Time delay LCMV beamformer

Description

The `TimeDelayLCMVBeamformer` object implements a time-delay linear constraint minimum variance beamformer.

To compute the beamformed signal:

1. Define and set up your time-delay LCMV beamformer. See "Construction" on page 1-1711.
2. Call `step` to perform the beamforming operation according to the properties of `phased.TimeDelayLCMVBeamformer`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.TimeDelayLCMVBeamformer` creates a time-delay linear constraint minimum variance (LCMV) beamformer System object, `H`. The object performs time delay LCMV beamforming on the received signal.

`H = phased.TimeDelayLCMVBeamformer(Name,Value)` creates a time-delay LCMV beamformer object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`SensorArray`

Handle to sensor array

Specify the sensor array as a handle. The sensor array must be an array object in the `phased` package. The array cannot contain subarrays.

Default: `phased.ULA` with default property values

`PropagationSpeed`

Signal propagation speed

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can specify this property as single or double precision.

Default: Speed of light

SampleRate

Signal sampling rate

Specify the signal sampling rate (in hertz) as a positive scalar. This property can be specified as single or double precision.

Default: `1e6`

FilterLength

FIR filter length

Specify the length of the FIR filter behind each sensor element in the array as a positive integer. This property can be specified as single or double precision.

Default: `2`

Constraint

Constraint matrix

Specify the constraint matrix used for time-delay LCMV beamformer as an *M*-by-*K* matrix. Each column of the matrix is a constraint and *M* is the number of degrees of freedom of the beamformer. For a time-delay LCMV beamformer, the number of degrees of freedom is the product of the number of elements of the array and the filter length specified by the value of the `FilterLength` property. This property can be specified as single or double precision.

Default: `[1;1]`

DesiredResponse

Desired response vector

Specify the desired response used for time-delay LCMV beamformer as a column vector of length *K*, where *K* is the number of constraints in the `Constraint` property. Each element in the vector defines the desired response of the constraint specified in the corresponding column of the `Constraint` property. This property can be specified as single or double precision.

Default: `1`, which is equivalent to a distortionless response

DiagonalLoadingFactor

Diagonal loading factor

Specify the diagonal loading factor as a positive scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample support is small. This property is tunable. This property can be specified as single or double precision.

Default: `0`

TrainingInputPort

Add input to specify training data

To specify additional training data, set this property to `true` and use the corresponding input argument when you invoke `step`. To use the input signal as the training data, set this property to `false`.

Default: `false`

DirectionSource

Source of beamforming direction

Specify whether the beamforming direction comes from the `Direction` property of this object or from an input argument in `step`. Values of this property are:

'Property'	The `Direction` property of this object specifies the beamforming direction.
'Input port'	An input argument in each invocation of `step` specifies the beamforming direction.

Default: `'Property'`

Direction

Beamforming direction

Specify the beamforming direction of the beamformer as a column vector of length 2. The direction is specified in the format of `[AzimuthAngle; ElevationAngle]` (in degrees). The azimuth angle is between –180° and 180°. The elevation angle is between –90° and 90°. This property applies when you set the `DirectionSource` property to `'Property'`. This property can be specified as single or double precision.

Default: `[0; 0]`

WeightsOutputPort

Output beamforming weights

To obtain the weights used in the beamformer, set this property to `true` and use the corresponding output argument when invoking `step`. If you do not want to obtain the weights, set this property to `false`.

Default: `false`

Methods

step	Perform time-delay LCMV beamforming

Common to All System Objects	
release	Allow System object property value changes

Examples

Time-Delay LCMV Beamformer

Apply a time delay LCMV beamformer to an 11-element acoustic ULA array. The elements are omnidirectional microphones. The incident angle of the signal is -50 degrees in azimuth and 30 degrees in elevation. The incident signal is an FM chirp with 500 Hz bandwidth. The propagation speed is a typical speed of sound in air, 340 m/s.

Simulate the signal and add noise.

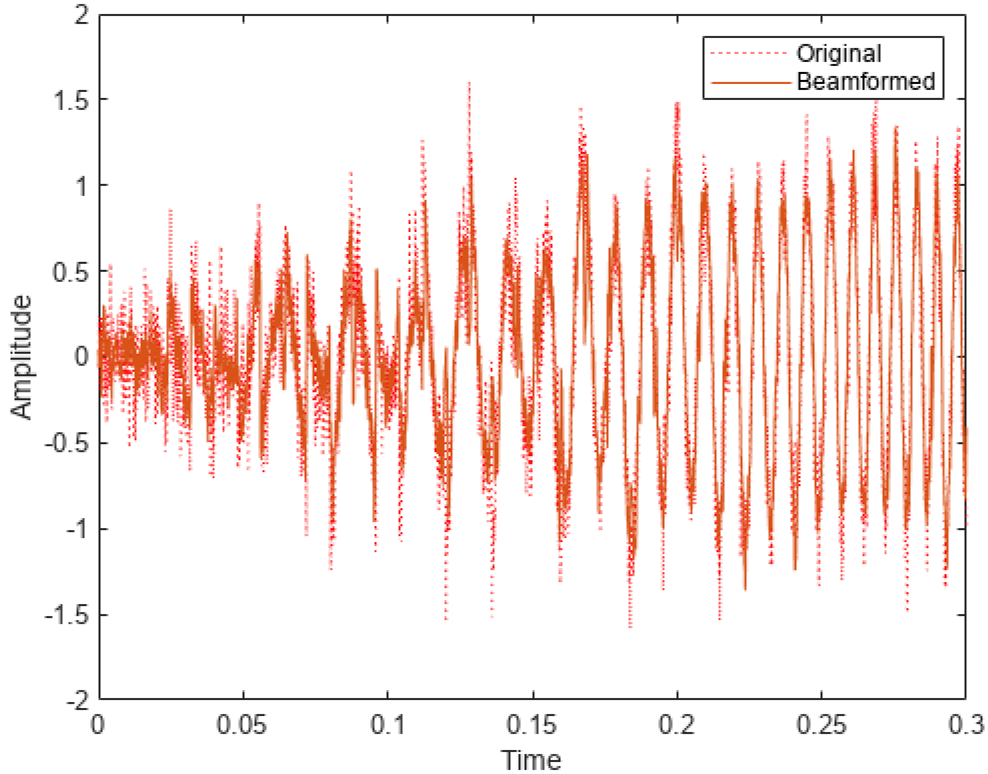
```
nElem = 11;
microphone = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
array = phased.ULA('Element',microphone,'NumElements',nElem,'ElementSpacing',0.04);
fs = 8000;
t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340;
collector = phased.WidebandCollector('Sensor',array,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',false);
incidentAngle = [-50;30];
x = collector(x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x + noise;
```

Create and apply the time-delay LCMV beamformer. Specify a filterlength of 5.

```
filterLength = 5;
constraintMatrix = kron(eye(filterLength),ones(nElem,1));
desiredResponseVector = eye(filterLength,1);
beamformer = phased.TimeDelayLCMVBeamformer('SensorArray',array,...
    'PropagationSpeed',c,'SampleRate',fs,'FilterLength',filterLength,...
    'Direction',incidentAngle,'Constraint',constraintMatrix,...
    'DesiredResponse',desiredResponseVector);
y = beamformer(rx);
```

Compare the beamformer output to the input to the middle sensor.

```
plot(t,rx(:,6),'r:',t,y)
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Algorithms

Beamforming Algorithms

The beamforming algorithm is the time-domain counterpart of the narrowband linear constraint minimum variance (LCMV) beamformer. The algorithm does the following:

1. Steers the array to the beamforming direction.
2. Applies an FIR filter to the output of each sensor to achieve the specified constraints. The filter is specific to each sensor.

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Frost, O. "An Algorithm For Linearly Constrained Adaptive Array Processing", *Proceedings of the IEEE*. Vol. 60, Number 8, August, 1972, pp. 926–935.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Requires dynamic memory allocation. See "Limitations for System Objects that Require Dynamic Memory Allocation".
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data `X` is single precision, the output data is single precision. If the input data `X` is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.FrostBeamformer` | `phased.PhaseShiftBeamformer` | `phased.SubbandPhaseShiftBeamformer` | `phased.TimeDelayBeamformer` | `uv2azel` | `phitheta2azel`

Topics

"Wideband Beamforming"

step

System object: `phased.TimeDelayLCMVBeamformer`
Package: `phased`

Perform time-delay LCMV beamforming

Syntax

```
Y = step(H,X)
Y = step(H,X,XT)
Y = step(H,X,ANG)
[Y,W] = step( ___ )
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` performs time-delay LCMV beamforming on the input, `X`, and returns the beamformed output in `Y`. `X` is an *M*-by-*N* matrix where *N* is the number of elements of the sensor array. *M* must be larger than the FIR filter length specified in the `FilterLength` property. `Y` is a column vector of length *M*.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`Y = step(H,X,XT)` uses `XT` as the training samples to calculate the beamforming weights when you set the `TrainingInputPort` property to `true`. `XT` is an *M*-by-*N* matrix where *N* is the number of elements of the sensor array. *M* must be larger than the FIR filter length specified in the `FilterLength` property.

`Y = step(H,X,ANG)` uses `ANG` as the beamforming direction, when you set the `DirectionSource` property to `'Input port'`. `ANG` is a column vector of length 2 in the form of `[AzimuthAngle; ElevationAngle]` (in degrees). The azimuth angle must be between –180° and 180°, and the elevation angle must be between –90° and 90°.

You can combine optional input arguments when their enabling properties are set: `Y = step(H,X,XT,ANG)`

`[Y,W] = step( ___ )` returns additional output, `W`, as the beamforming weights when you set the `WeightsOutputPort` property to `true`. `W` is a column vector of length *L*, where *L* is the number of degrees of freedom of the beamformer. For a time-delay LCMV beamformer, the number of degrees of freedom is given by the product of the number of elements of the array and the filter length specified by the value of the `FilterLength` property.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of

the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

All input and output arguments can be single or double precision.

Examples

Time-Delay LCMV Beamformer

Apply a time delay LCMV beamformer to an 11-element acoustic ULA array. The elements are omnidirectional microphones. The incident angle of the signal is -50 degrees in azimuth and 30 degrees in elevation. The incident signal is an FM chirp with 500 Hz bandwidth. The propagation speed is a typical speed of sound in air, 340 m/s.

Simulate the signal and add noise.

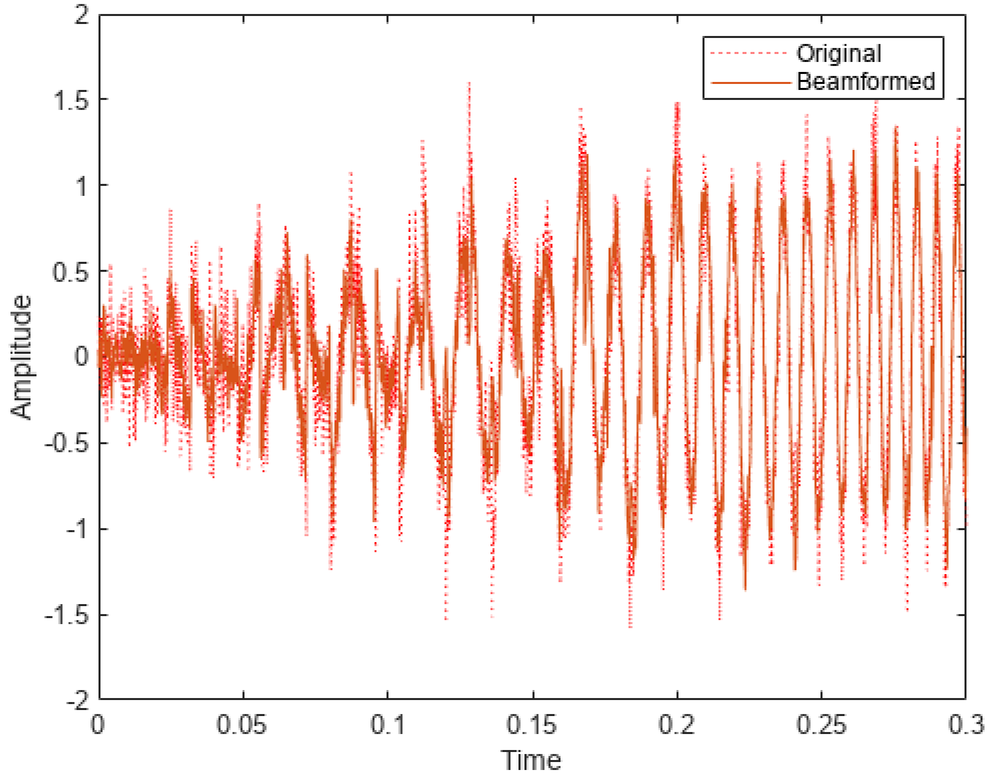
```
nElem = 11;
microphone = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
array = phased.ULA('Element',microphone,'NumElements',nElem,'ElementSpacing',0.04);
fs = 8000;
t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340;
collector = phased.WidebandCollector('Sensor',array,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',false);
incidentAngle = [-50;30];
x = collector(x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x + noise;
```

Create and apply the time-delay LCMV beamformer. Specify a filterlength of 5.

```
filterLength = 5;
constraintMatrix = kron(eye(filterLength),ones(nElem,1));
desiredResponseVector = eye(filterLength,1);
beamformer = phased.TimeDelayLCMVBeamformer('SensorArray',array,...
    'PropagationSpeed',c,'SampleRate',fs,'FilterLength',filterLength,...
    'Direction',incidentAngle,'Constraint',constraintMatrix,...
    'DesiredResponse',desiredResponseVector);
y = beamformer(rx);
```

Compare the beamformer output to the input to the middle sensor.

```
plot(t,rx(:,6),'r:',t,y)
xlabel('Time')
ylabel('Amplitude')
legend('Original','Beamformed')
```



Algorithms

The beamforming algorithm is the time-domain counterpart of the narrowband linear constraint minimum variance (LCMV) beamformer. The algorithm does the following:

1. Steers the array to the beamforming direction.
2. Applies an FIR filter to the output of each sensor to achieve the specified constraints. The filter is specific to each sensor.

See Also

`uv2azel` | `phitheta2azel`

phased.TimeVaryingGain

Package: `phased`

Time varying gain control

Description

The `TimeVaryingGain` object applies a time varying gain to input signals. Time varying gain (TVG) is sometimes called automatic gain control (AGC).

To apply the time varying gain to the signal:

1. Define and set up your time varying gain controller. See "Construction" on page 1-1720.
2. Call `step` to apply the time varying gain according to the properties of `phased.TimeVaryingGain`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.TimeVaryingGain` creates a time varying gain control System object, `H`. The object applies a time varying gain to the input signal to compensate for the signal power loss due to the range.

`H = phased.TimeVaryingGain(Name,Value)` creates an object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`RangeLossSource`

Source of range losses

Specify the source of range losses as either `'Property'` or `'Input port'`. When you specify `RangeLossSource` as `'Property'`, the range loss for each sample is set in the `RangeLoss` property. When you specify the `RangeLossSource` as `'Input port'`, the range losses are specified using an input argument to the `step` method.

Default: `'Property'`

`RangeLoss`

Loss at each input sample range

Specify the loss due to range as a vector — elements correspond to the samples in the input signal. Units are in dB. This property can have single or double precision.

Default: `0`

`ReferenceLoss`

Loss at reference range

Specify the loss at a given reference range as a scalar. Units are in dB. This property can have single or double precision.

Default: `0`

Methods

step	Apply time varying gains to input signal

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Apply Time Varying Gain to Adjust for Range Loss

Apply time varying gain to a signal to compensate for signal power loss due to range.

First, create a signal with range loss. Set the reference loss to 16 dB.

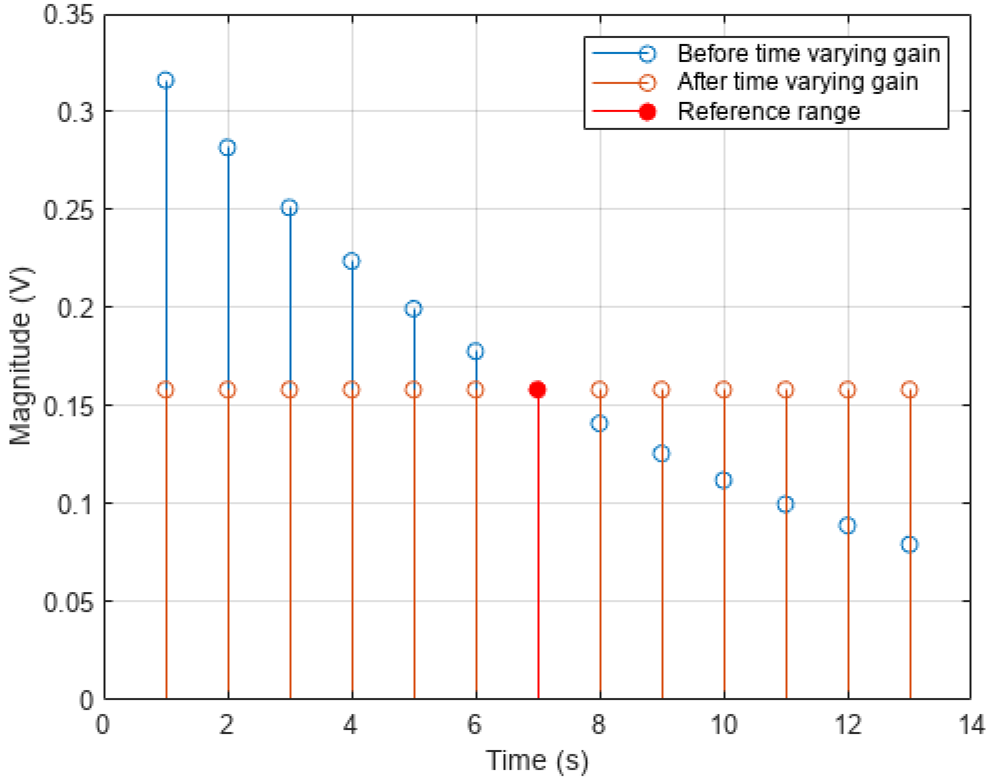
```
rngloss = 10:22;
refloss = 16;
t = (1:length(rngloss))';
x = 1./db2mag(rngloss(:));
```

Then add gain to compensate for range loss.

```
gain = phased.TimeVaryingGain('RangeLoss',rngloss,'ReferenceLoss',refloss);
y = gain(x);
```

Plot the signal with loss and the compensated signal.

```
tref = find(rngloss==refloss);
stem([t t],[abs(x) abs(y)])
hold on
stem(tref,x(tref),'filled','r')
xlabel('Time (s)'); ylabel('Magnitude (V)')
grid on
legend('Before time varying gain','After time varying gain',...
    'Reference range')
```



Algorithms

Data Precision

This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

Version History

Introduced in R2011a

References

[1] Edde, B. *Radar: Principles, Technology, Applications*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).
- This System object supports single and double precision for input data, properties, and arguments. If the input data X is single precision, the output data is single precision. If the input data X is double precision, the output data is double precision. The precision of the output is independent of the precision of the properties and other arguments.

See Also

`phased.MatchedFilter` | `pulsint`

step

System object: phased.TimeVaryingGain
Package: phased

Apply time varying gains to input signal

Syntax

```
Y = step(H,X)
Y = step(H,X,L)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` applies time varying gains to the input signal matrix `X`. The process equalizes power levels across all samples to match a given reference range. The compensated signal is returned in `Y`. `X` can be a column vector, a matrix, or a cube. The gain is applied to each column in `X` independently. The number of rows in `X` cannot exceed the length of the loss vector specified in the `RangeLoss` property. `Y` has the same dimensionality as `X`. `X` can be single or double precision.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`Y = step(H,X,L)` in addition, specifies the range loss, `L` as a columns vector. Use this argument only when you set the `RangeLossSource` property to `'Input port'`. The length of `L` must be equal to or greater than the number of rows of `X`. `L` can be single or double precision.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Apply Time Varying Gain to Adjust for Range Loss

Apply time varying gain to a signal to compensate for signal power loss due to range.

First, create a signal with range loss. Set the reference loss to 16 dB.

```
rngloss = 10:22;
refloss = 16;
```

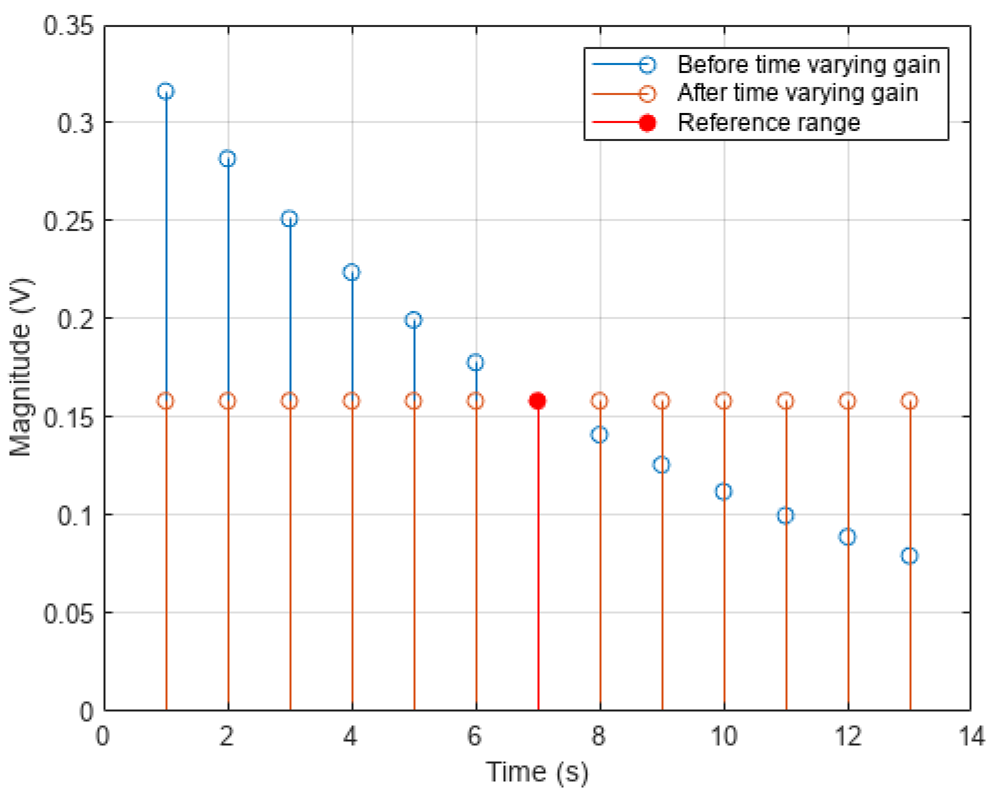
```
t = (1:length(rngloss))';
x = 1./db2mag(rngloss(:));
```

Then add gain to compensate for range loss.

```
gain = phased.TimeVaryingGain('RangeLoss',rngloss,'ReferenceLoss',refloss);
y = gain(x);
```

Plot the signal with loss and the compensated signal.

```
tref = find(rngloss==refloss);
stem([t t],[abs(x) abs(y)])
hold on
stem(tref,x(tref),'filled','r')
xlabel('Time (s)'); ylabel('Magnitude (V)')
grid on
legend('Before time varying gain','After time varying gain',...
    'Reference range')
```



phased.Transmitter

Package: `phased`

Transmitter

Description

The `Transmitter` object implements a waveform transmitter.

To compute the transmitted signal:

1. Define and set up your waveform transmitter. See “Construction” on page 1-1726.
2. Call `step` to compute the transmitted signal according to the properties of `phased.Transmitter`. The behavior of `step` is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`H = phased.Transmitter` creates a transmitter System object, `H`. This object transmits the input waveform samples with specified peak power.

`H = phased.Transmitter(Name,Value)` creates a transmitter object, `H`, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`).

Properties

`PeakPower`

Peak power

Specify the transmit peak power (in watts) as a positive scalar.

Default: `5000`

`Gain`

Transmit gain

Specify the transmit gain (in decibels) as a real scalar.

Default: `20`

`LossFactor`

Loss factor

Specify the transmit loss factor (in decibels) as a nonnegative scalar.

Default: `0`

InUseOutputPort

Enable transmitter status output

To obtain the transmitter in-use status for each output sample, set this property to `true` and use the corresponding output argument when invoking `step`. In this case, 1's indicate the transmitter is on, and 0's indicate the transmitter is off. If you do not want to obtain the transmitter in-use status, set this property to `false`.

Default: `false`

CoherentOnTransmit

Preserve coherence among pulses

Specify whether to preserve coherence among transmitted pulses. When you set this property to `true`, the transmitter does not introduce any random phase to the output pulses. When you set this property to `false`, the transmitter adds a random phase noise to each transmitted pulse. The random phase noise is introduced by multiplication of the pulse by $e^{j\phi}$where ϕ is a uniform random variable on the interval [0,2π].

Default: `true`

PhaseNoiseOutputPort

Enable pulse phase noise output

To obtain the introduced transmitter random phase noise for each output sample, set this property to `true` and use the corresponding output argument when invoking `step`. You can use in the receiver to simulate coherent on receive systems. If you do not want to obtain the random phase noise, set this property to `false`. This property applies when you set the `CoherentOnTransmit` property to `false`.

Default: `false`

SeedSource

Source of seed for random number generator

`'Auto'`	The default MATLAB random number generator produces the random numbers. Use `'Auto'` if you are using this object with Parallel Computing Toolbox software.
`'Property'`	The object uses its own private random number generator to produce random numbers. The `Seed` property of this object specifies the seed of the random number generator. Use `'Property'` if you want repeatable results and are not using this object with Parallel Computing Toolbox software.

This property applies when you set the `CoherentOnTransmit` property to `false`.

Default: `'Auto'`

Seed

Seed for random number generator

Specify the seed for the random number generator as a scalar integer between 0 and 2^{32}–1. This property applies when you set the `CoherentOnTransmit` property to `false` and the `SeedSource` property to `'Property'`.

Default: `0`

Methods

reset	Reset states of transmitter object
step	Transmit pulses

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Transmit LFM Pulse

Transmit a pulse containing a linear FM waveform with a bandwidth of 5 MHz. The sample rate is 10 MHz and the pulse repetition frequency is 10 kHz.

```
fs = 1e7;
waveform = phased.LinearFMWaveform('SampleRate',fs, ...
    'PulseWidth',1e-5,'SweepBandwidth',5e6);
x = waveform();
transmitter = phased.Transmitter('PeakPower',5e3);
y = transmitter(x);
```

Version History

Introduced in R2011a

References

[1] Edde, B. *Radar: Principles, Technology, Applications*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[3] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.Radiator` | `phased.ReceiverPreamp`

reset

System object: `phased.Transmitter`
Package: `phased`

Reset states of transmitter object

Syntax

`reset(H)`

Description

`reset(H)` resets the states of the `Transmitter` object, `H`. This method resets the random number generator state if the `SeedSource` property is applicable and has the value `'Property'`.

step

System object: `phased.Transmitter`
Package: `phased`

Transmit pulses

Syntax

```
Y = step(H,X)
[Y,STATUS] = step(H,X)
[Y,PHNOISE] = step(H,X)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X)` returns the transmitted signal `Y`, based on the input waveform `X`. `Y` is the amplified `X` where the amplification is based on the characteristics of the transmitter, such as the peak power and the gain.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

`[Y,STATUS] = step(H,X)` returns additional output `STATUS` as the on/off status of the transmitter when the `InUseOutputPort` property is `true`. `STATUS` is a logical vector where `true` indicates the transmitter is on for the corresponding sample time, and `false` indicates the transmitter is off.

`[Y,PHNOISE] = step(H,X)` returns the additional output `PHNOISE` as the random phase noise added to each transmitted sample when the `CoherentOnTransmit` property is `false` and the `PhaseNoiseOutputPort` property is `true`. `PHNOISE` is a vector which has the same dimension as Y. Each element in `PHNOISE` contains the random phase between 0 and 2*pi, added to the corresponding sample in `Y` by the transmitter.

You can combine optional output arguments when their enabling properties are set. Optional outputs must be listed in the same order as the order of the enabling properties. For example:

```
[Y,STATUS,PHNOISE] = step(H,X)
```

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Examples

Transmit LFM Pulse

Transmit a pulse containing a linear FM waveform with a bandwidth of 5 MHz. The sample rate is 10 MHz and the pulse repetition frequency is 10 kHz.

```
fs = 1e7;
waveform = phased.LinearFMWaveform('SampleRate',fs, ...
    'PulseWidth',1e-5,'SweepBandwidth',5e6);
x = waveform();
transmitter = phased.Transmitter('PeakPower',5e3);
y = transmitter(x);
```

phased.UCA

Package: phased

Uniform circular array

Description

The phased.UCA System object creates a uniform circular array (UCA). A UCA is formed from identical sensor elements equally spaced around a circle.

To compute the response for the array for specified directions:

1. Define and set up your uniform circular array. See "Construction" on page 1-1733.
2. Call step to compute the response according to the properties of phased.UCA. The behavior of step is specific to each object in the toolbox.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

sUCA = phased.UCA creates a uniform circular array (UCA) System object, sUCA, consisting of five identical isotropic antenna elements,phased.IsotropicAntennaElement. The elements are equally spaced around a circle of radius 0.5 meters.

sUCA = phased.UCA(Name,Value) creates a System object, sUCA, with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

sUCA = phased.UCA(N,R) creates a UCA System object, sUCA, with the NumElements property set to N and the Radius property set to R. This syntax creates a UCA consisting of isotropic antenna elements, phased.IsotropicAntennaElement.

sUCA = phased.UCA(N,R,Name,Value) creates a UCA System object, sUCA, with the NumElements property set to N, the Radius property set to R, and other specified property Names set to the specified Values.

Properties

Element — phased array element

isotropic antenna element System object with default properties (default) | Phased Array System Toolbox antenna, microphone, or transducer element System object | Antenna Toolbox antenna System object

Phased array element, specified as a Phased Array System Toolbox antenna, microphone, or transducer element or Antenna Toolbox antenna.

Example: phased.CosineAntennaElement

`NumElements` — Number of array elements
5 (default) | integer greater than one

Number of array elements, specified as an integer greater than one.

Example: 3

`Radius` — Array radius
0.5 (default) | positive scalar

Array radius, specified as a positive scalar in meters.

Example: `2.5`

`ArrayNormal` — Array normal direction
`'z'` (default) | `'x'` | `'y'`

Array normal direction, specified as one of `'x'`, `'y'`, or `'z'`. UCA elements lie in a plane orthogonal to the array normal direction. Element boresight vectors lie in the same plane and point radially outward from the origin.

ArrayNormal Property Value	Element Positions and Boresight Directions
`'x'`	Array elements lie on the *yz*-plane. All element boresight vectors lie in the *yz*-plane and point outward from the array center.
`'y'`	Array elements lie on the *zx*-plane. All element boresight vectors lie in the *zx*-plane and point outward from the array center.
`'z'`	Array elements lie on the *xy*-plane. All element boresight vectors lie in the *xy*-plane and point outward from the array center.

Example: `'y'`

`Taper` — Element tapering
1 (default) | complex-valued scalar | complex-valued 1-by-*N* row vector | complex-valued *N*-by-1 column vector

Element tapering or weighting, specified as a complex-valued scalar, 1-by-*N* row vector, or *N*-by-1 column vector. The quantity *N* represents the number of elements of the array. Tapers, also known as weights, are applied to each sensor element in the sensor array and modify both the amplitude and phase of the received data. If `'Taper'` is a scalar, the same taper value is applied to all element. If `'Taper'` is a vector, each taper value is applied to the corresponding sensor element.

Example: `[1 2 3 2 1]`

Methods

Specific to `phased.UCA` Object	
`beamwidth`	Compute and display beamwidth of an array
`collectPlaneWave`	Simulate received plane waves

Specific to phased.UCA Object	
directivity	Directivity of uniform circular array
getElementNormal	Normal vectors for array elements
getElementPosition	Positions of array elements
getElementSpacing	Spacing between array elements
getNumElements	Number of elements in array
getTaper	Array element tapers
isPolarizationCapable	Polarization capability
pattern	Plot UCA array pattern
patternAzimuth	Plot UCA array directivity or pattern versus azimuth
patternElevation	Plot UCA array directivity or pattern versus elevation
perturbations	Perturbations defined on phased array
perturbedArray	Apply perturbations to phased array
perturbedPattern	Compute and plot azimuth pattern of perturbed array
step	Output responses of array elements
viewArray	View array geometry

Common to All System Objects	
release	Allow System object property value changes

Examples

Pattern of 11-Element UCA Antenna Array

Create an 11-element uniform circular array (UCA) having a 1.5 m radius and operating at 500 MHz. The array consists of short-dipole antenna elements. First, display the vertical component of the response at 45 degrees azimuth and 0 degrees elevation. Then plot the azimuth and elevation directivities.

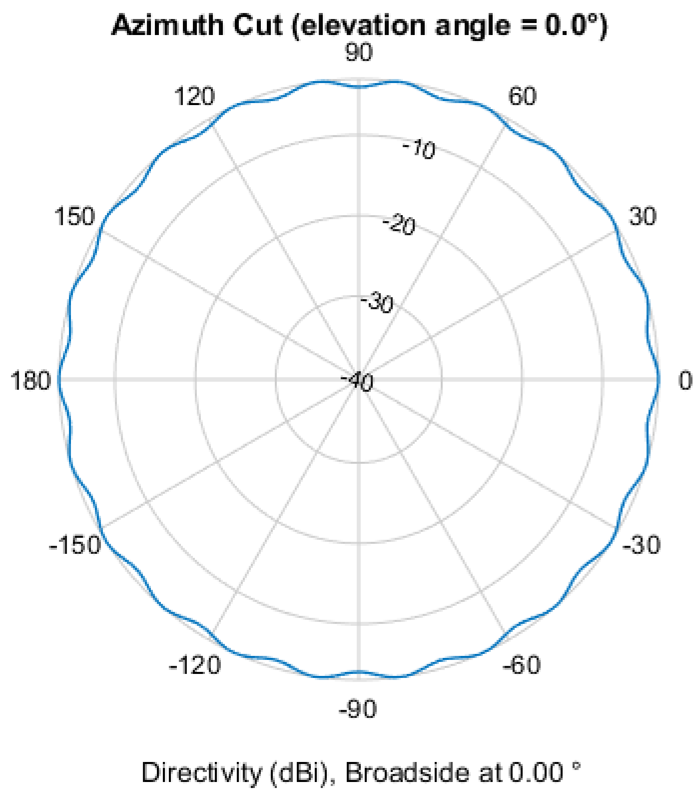
```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[50e6,1000e6],...
    'AxisDirection','Z');
array = phased.UCA('NumElements',11,'Radius',1.5,'Element',antenna);
```

```
fc = 500e6;
ang = [45;0];
resp = array(fc,ang);
disp(resp.V)
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

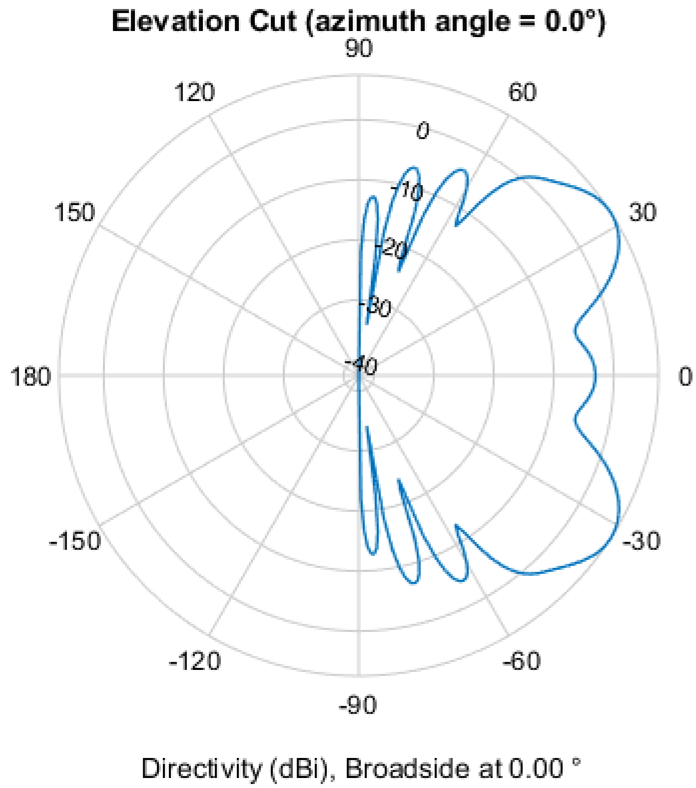
Display the azimuth directivity pattern at 500 MHz for azimuth angles between -180 and 180 degrees.

```
c = physconst('LightSpeed');
pattern(array,fc,[-180:180],0,'Type','directivity','PropagationSpeed',c)
```



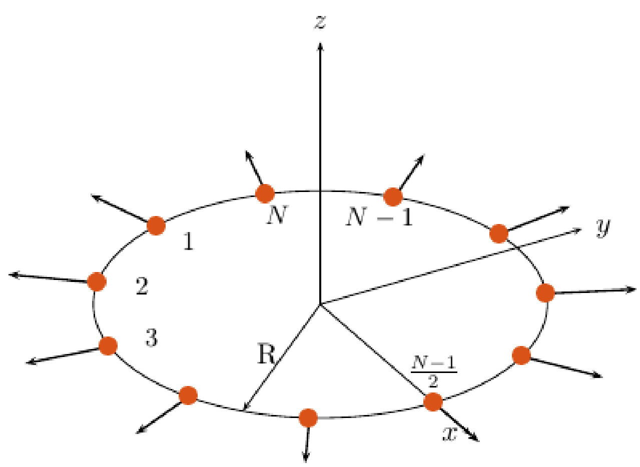
Display the elevation directivity pattern at 500 MHz for elevation angles between -90 and 90 degrees.

```
pattern(array,fc,[0],[-90:90],'Type','directivity','PropagationSpeed',c)
```



Algorithms

A UCA is formed from *N* identical sensor elements equally spaced around a circle of radius *R*. The circle lies in the *xy*-plane of the local coordinate system whose origin lies at the center of the circle. The positions of the elements are defined with respect to the local array coordinate system. The circular array lies in the *xy*-plane of the coordinate system. The normal to the UCA plane lies along the positive *z*-axis. The elements are oriented so that their main response directions (normals) point radially outward in the *xy*-plane.



If the number of elements of the array is odd, the middle element lies on the *x*-axis. If the number of elements is even, the midpoint between the two middle elements lies on the *x*-axis. For an array of *N* elements, the azimuth angle of the position of the *nth* element is given by

$$\varphi_n = (-(N-1)/2 + n - 1) \cdot 360/N \quad n = 1, \ldots, N$$

The azimuth angle is defined as the angle, in the *xy*-plane, from the *x*-axis toward the *y*-axis. The elevation angle is defined as the angle from the *xy*-plane toward the *z*-axis. The angular distance between any two adjacent elements is *360/N* degrees. Azimuth angle values are in degrees. Elevation angles for all array elements are zero.

Version History

Introduced in R2015a

References

[1] Brookner, E., ed. *Radar Technology*. Lexington, MA: LexBook, 1996.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002, pp. 274–304.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, `plotResponse`, and `viewArray` methods are not supported.

- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.ULA` | `phased.URA` | `phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CrossedDipoleAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ShortDipoleAntennaElement` | `phased.CustomMicrophoneElement` | `phased.OmnidirectionalMicrophoneElement`

Topics

“Phased Array Gallery”

directivity

System object: phased.UCA
Package: phased

Directivity of uniform circular array

Syntax

```
D = directivity(sArray,FREQ,ANGLE)
D = directivity(sArray,FREQ,ANGLE,Name,Value)
```

Description

`D = directivity(sArray,FREQ,ANGLE)` returns the "Directivity (dBi)" on page 1-1743 of a uniform circular array (UCA) of antenna or microphone elements, `sArray`, at frequencies specified by `FREQ` and in angles of direction specified by `ANGLE`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`D = directivity(sArray,FREQ,ANGLE,Name,Value)` returns the directivity with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

`sArray` — Uniform circular array

System object

Uniform circular array, specified as a `phased.UCA` System object.

Example: `sArray= phased.UCA;`

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

ANGLE — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If ANGLE is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, [az;el]. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If ANGLE is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: [45 60; 0 10]

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

PropagationSpeed — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of 'PropagationSpeed' and a positive scalar in meters per second.

Example: 'PropagationSpeed',physconst('LightSpeed')

Data Types: double

Weights — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of 'Weights' and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by FREQ.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of 'Weights' for the corresponding frequency in FREQ.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of UCA

Compute the directivity of two uniform circular arrays (UCA) at zero degrees azimuth and elevation. The first array consists of isotropic antenna elements. The second array consists of cosine antenna elements. In addition, compute the directivity of the cosine element array steered to a 45 degrees elevation.

Array of isotropic antenna elements

First, create a 10-element UCA with a radius of one-half meter consisting of isotropic antenna elements. Set the signal frequency to 300 MHz.

```
c = physconst('LightSpeed');
fc = 300e6;
sIso = phased.IsotropicAntennaElement;
sArray = phased.UCA('Element',sIso,'NumElements',10,'Radius',0.5);
ang = [0;0];
d = directivity(sArray,fc,ang,'PropagationSpeed',c)
```

```
d = -1.1423
```

Array of cosine antenna elements

Next, create a 10-element UCA of cosine antenna elements also with a 0.5 meter radius.

```
sCos = phased.CosineAntennaElement('CosinePower',[3,3]);
sArray1 = phased.UCA('Element',sCos,'NumElements',10,'Radius',0.5);
ang = [0;0];
d = directivity(sArray1,fc,ang,'PropagationSpeed',c)
```

```
d = 3.2550
```

The directivity is increased due to the added directivity of the cosine antenna elements.

Steered array of cosine antenna elements

Finally, steer the cosine antenna array toward 45 degrees elevation, and then examine the directivity at 45 degrees.

```
ang = [0;45];
lambda = c/fc;
w = steervec(getElementPosition(sArray1)/lambda,ang);
d = directivity(sArray1,fc,ang,'PropagationSpeed',c,...
    'Weights',w)
```

```
d = -3.1410
```

The directivity is decreased because of the combined reduction of directivity of the elements and the array.

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth` | `patternElevation`

collectPlaneWave

System object: `phased.UCA`
Package: `phased`

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

`Y = collectPlaneWave(H,X,ANG)` returns the received signals at the sensor array, `H`, when the input signals indicated by `X` arrive at the array from the directions specified in `ANG`.

`Y = collectPlaneWave(H,X,ANG,FREQ)`, in addition, specifies the incoming signal carrier frequency in `FREQ`.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)`, in addition, specifies the signal propagation speed in `C`.

Input Arguments

`H` — Uniform circular array

System object

Uniform circular array specified as a `phased.UCA` System object.

Example: `H = phased.UCA();`

`X` — Incoming signals

M-column matrix

Incoming signals, specified as an *M*-column matrix. Each column of `X` represents an individual incoming signal.

Example: `[1,5;2,10;3,10]`

Data Types: `double`
Complex Number Support: Yes

`ANG` — Arrival directions of incoming signals

1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Arrival directions of incoming signals, specified as a 1-by-*M* vector or a 2-by-*M* matrix, where *M* is the number of incoming signals. Each column specifies the direction of arrival of the corresponding signal in `X`. If `ANG` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the incoming signal `[az;el]`. Angular units are in degrees. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°.

If ANG is a 1-by-*M* vector, then each entry represents a set of azimuth angles, with the elevation angles assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the arrival direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, the azimuth angle is positive.

The elevation angle is the angle between the arrival direction vector and the *xy*-plane. When measured toward the *z* axis, the elevation angle is positive.

Example: `[20,30;15,25]`

Data Types: `double`

FREQ — Signal carrier frequency
3e8 (default) | positive scalar

Signal carrier frequency, specified as a positive scalar in hertz.

Data Types: `double`

C — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar in meters per second.

Example: `physconst('LightSpeed')`

Data Types: `double`

Output Arguments

Y — Received signals
N-column complex-valued row vector

Received signals, returned as an *N*-column complex-valued row vector. The quantity *N* is the number of elements in the array. Each column of Y contains the combined received signals at the corresponding array element.

Examples

Simulate Received Signal at 5-Element UCA

Create a random signal arriving at a 5-element UCA from 10 degrees azimuth and 30 degrees azimuth. Both signals have an elevation angle of 0 degrees. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz. The signals are two random noise signals of three samples each.

```
sUCA = phased.UCA('NumElements',5,'Radius',2.0);
y = collectPlaneWave(sUCA,randn(3,2),[10 30],100e6,...
    physconst('LightSpeed'));
disp(y)
```

```
  -0.8817 + 1.0528i   1.0037 - 0.3636i  -1.0579 - 0.8531i  -1.0698 + 0.5187i  -0.6388 - 0.9769i
  -1.6512 + 1.3471i   1.7358 + 0.7662i  -1.2932 - 1.6792i  -1.0279 + 1.6997i  -1.8283 - 0.7336i
   2.5071 - 2.4424i  -2.7270 - 0.2435i   2.4009 + 2.4977i   2.1808 - 2.1178i   2.3743 + 1.8105i
```

Algorithms

`collectPlaneWave` modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. The method does not account for the response of individual elements in the array.

For further details, see [1].

Version History

Introduced in R2015a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

getElementNormal

System object: `phased.UCA`
Package: `phased`

Normal vectors for array elements

Syntax

```
normvec = getElementNormal(sArray)
normvec = getElementNormal(sArray,elemidx)
```

Description

`normvec = getElementNormal(sArray)` returns the element normals of the `phased.UCA` System object, `sArray`. `normv` is a 2-by-*N* matrix, where *N* is the number of elements in `sArray`. Each column of `normv` specifies the normal direction of the corresponding element in the local coordinate system in the form `[azimuth;elevation]`. Units are degrees. For details regarding the local coordinate system of a UCA, type

```
phased.UCA.coordinateSystemInfo;
```

at the command line.

`normvec = getElementNormal(sArray,elemidx)` returns only the normals of the elements that are specified in the element index vector `elemidx`.

Input Arguments

`sArray` — Uniform circular array
`phased.UCA` System object

Uniform circular array, specified as a `phased.UCA` System object.

Example: `phased.UCA`

`elemidx` — Element index vector
all elements (default) | vector of positive integers

Element index vector, specified as a vector of positive integers each of which takes a value from 1 to *N*. The dimension *N* is the number of elements of the array.

Example: `[1,2,3]`

Output Arguments

`normvec` — Normal vector
2-by-*M* real-valued matrix

Normal vector of array elements, returned as a 2-by-*M* real matrix. Each column of `normvec` specifies the normal direction of the corresponding element in the local coordinate system in the form `[azimuth;elevation]`. Units are degrees. If the input argument `elemidx` is not specified, *M* is the number of elements of the array, *N*. If `elemidx` is specified, *M* is the dimension of `elemidx`.

Examples

UCA Element Normal Vectors

Construct three different 7-element UCA with a radius of 0.5 meters, and obtain the normal vectors of the middle three elements. Choose the array normal vectors to point along the *x*-, *y*-, and *z*-axes.

First, choose the array normal along the *x*-axis.

```
sUCA1 = phased.UCA('NumElements',7,'Radius',0.5,'ArrayNormal','x');
pos = getElementPosition(sUCA1,[3,4,5])
```

```
pos = 3×3

         0         0         0
    0.3117    0.5000    0.3117
   -0.3909         0    0.3909
```

```
normvec = getElementNormal(sUCA1,[3,4,5])
```

```
normvec = 2×3

   90.0000   90.0000   90.0000
  -51.4286         0   51.4286
```

These outputs show that the array elements lie in the *yz*-plane. The normal vectors of the array elements also lie in the *yz*-plane and point outward like spokes on a wheel.

Next, choose the array normal along the *y*-axis.

```
sUCA2 = phased.UCA('NumElements',7,'Radius',0.5,'ArrayNormal','y');
pos = getElementPosition(sUCA2,[3,4,5])
```

```
pos = 3×3

    0.3117    0.5000    0.3117
         0         0         0
   -0.3909         0    0.3909
```

```
normvec = getElementNormal(sUCA2,[3,4,5])
```

```
normvec = 2×3

         0         0         0
  -51.4286         0   51.4286
```

These outputs show that the array elements lie in the *zx*-plane. The normal vectors of the array elements also lie in the *zx*-plane and also point outward.

Finally, set the array normal along the *z*-axis. This is the default value of array normal.

```
sUCA3 = phased.UCA('NumElements',7,'Radius',0.5,'ArrayNormal','z');
pos = getElementPosition(sUCA3,[3,4,5])
```

```
pos = 3×3

    0.3117    0.5000    0.3117
   -0.3909         0    0.3909
         0         0         0
```

```
normvec = getElementNormal(sUCA3,[3,4,5])
```

```
normvec = 2×3

  -51.4286         0   51.4286
         0         0         0
```

These outputs show that the array elements lie in the *xy*-plane. The normal vectors of the array elements also lie in the *xy*-plane and also point outward.

Version History

Introduced in R2015a

getElementPosition

System object: phased.UCA
Package: phased

Positions of array elements

Syntax

```
pos = getElementPosition(sUCA)
pos = getElementPosition(sUCA,elemidx)
```

Description

`pos = getElementPosition(sUCA)` returns the element positions of the `phased.UCA` System object, `sUCA`. `pos` is a 3-by-*N* matrix, where *N* is the number of elements in `sUCA`. Each column of `pos` defines the position of an element in the local coordinate system, in meters, using the form `[x;y;z]`. The origin of the local coordinate system is the center of the circular array.

`pos = getElementPosition(sUCA,elemidx)` returns only the positions of the elements that are specified in the element index vector `elemidx`.

Input Arguments

`sUCA` — Uniform circular array
`phased.UCA` System object

Uniform circular array, specified as a `phased.UCA` System object.

Example: `phased.UCA`

`elemidx` — Element index vector
all elements (default) | vector of positive integers

Element index vector, specified as a vector of positive integers each of which takes a value from 1 to *N*. The quantity *N* is the number of elements of the array.

Example: `[1,2,3]`

Output Arguments

`pos` — Positions of array elements
3-by-*M* real matrix

Positions of array elements, returned as a 3-by-*M* real matrix. If the input argument `elemidx` is not specified, *M* is the number of elements of the array, *N*. If `elemidx` is specified, *M* is the dimension of `elemidx`.

Examples

Positions of UCA Elements

Construct a 7-element UCA with a radius of 0.5 meters, and obtain the positions of the middle three elements.

```
sArray = phased.UCA('NumElements',7,'Radius',0.5);
pos = getElementPosition(sArray,[3,4,5])
```

```
pos = 3×3

    0.3117    0.5000    0.3117
   -0.3909         0    0.3909
         0         0         0
```

The output verifies that the position of the middle element of an array with an odd number of elements lies on the x-axis.

Version History

Introduced in R2015a

getElementSpacing

System object: `phased.UCA`
Package: `phased`

Spacing between array elements

Syntax

```
dist = getElementSpacing(sArray)
dist = getElementSpacing(sArray,disttype)
```

Description

`dist = getElementSpacing(sArray)` returns the arc length between adjacent elements of the `phased.UCA` System object, `sArray`.

`dist = getElementSpacing(sArray,disttype)` returns either the arc length or chord length between adjacent elements depending on the specification of `disttype`.

Input Arguments

`sArray` — Uniform circular array

`phased.UCA` System object

Uniform circular array, specified as a `phased.UCA` System object.

Example: `phased.UCA()`

`disttype` — Distance type

`'arc'` (default) | `'chord'`

Distance type to define path between adjacent array elements, specified as a either `'arc'` or `'chord'`. If `disttype` is specified as `'arc'`, the returned distance is the arc length between adjacent elements. If `disttype` is specified as `'chord'`, the returned distance is the chord length between adjacent elements.

Example: `'chord'`

Output Arguments

`spacing` — Spacing between elements

scalar

Spacing between elements, returned as a scalar. A uniform circular array has a unique distance between all pairs of adjacent elements. The distance depends only upon the radius of the array, R, and the angle between two adjacent elements, $\Delta\varphi$. The angle between two adjacent elements is computed from the number of elements, $\Delta\varphi = 2\pi/N$. If `disttype` is specified as `'arc'`, the method returns

$R\Delta\varphi$.

If `disttype` is specified as `'chord'`, the method returns

2Rsin(Δφ/2).

The chord distance is always less than the arc distance.

Examples

Spacing Between UCA Elements

Construct a 10-element UCA with a radius of 1.5 meters, and obtain the arc distance between any two adjacent elements. Then, obtain the chord distance.

```
sArray = phased.UCA('NumElements',10,'Radius',1.5);
dist = getElementSpacing(sArray,'arc')
```

```
dist = 0.9425
```

```
dist = getElementSpacing(sArray,'chord')
```

```
dist = 0.9271
```

Version History

Introduced in R2015a

getNumElements

System object: phased.UCA
Package: phased

Number of elements in array

Syntax

```
N = getNumElements(H)
```

Description

`N = getNumElements(H)` returns the number of elements, *N*, in the UCA object `H`.

Input Arguments

H — Uniform circular array
phased.UCA System object

Uniform circular array, specified as a `phased.UCA` System object.

Example: `H = phased.UCA();`

Output Arguments

N — Number of elements
positive integer

Number of elements of array, returned as a positive integer.

Examples

Number of Elements of UCA

Create a UCA with the default number of elements. Verify that there are five elements.

```
sArray = phased.UCA();
N = getNumElements(sArray)
```

```
N = 5
```

Version History

Introduced in R2015a

getTaper

System object: phased.UCA
Package: phased

Array element tapers

Syntax

```
WTS = getTaper(H)
```

Description

`WTS = getTaper(H)` returns the tapers, `WTS`, applied to each element of the phased uniform circular array (UCA), `H`. Tapers are often referred to as weights.

Input Arguments

H — Uniform circular array

System object

Uniform circular array, specified as a `phased.ULA` System object.

Example: `H = phased.UCA();`

Output Arguments

WTS — Array element tapers

N-by-1 complex-valued vector

Array element tapers, returned as an *N*-by-1 complex-valued vector, where *N* is the number of elements in the array.

Examples

Show UCA Element Tapers

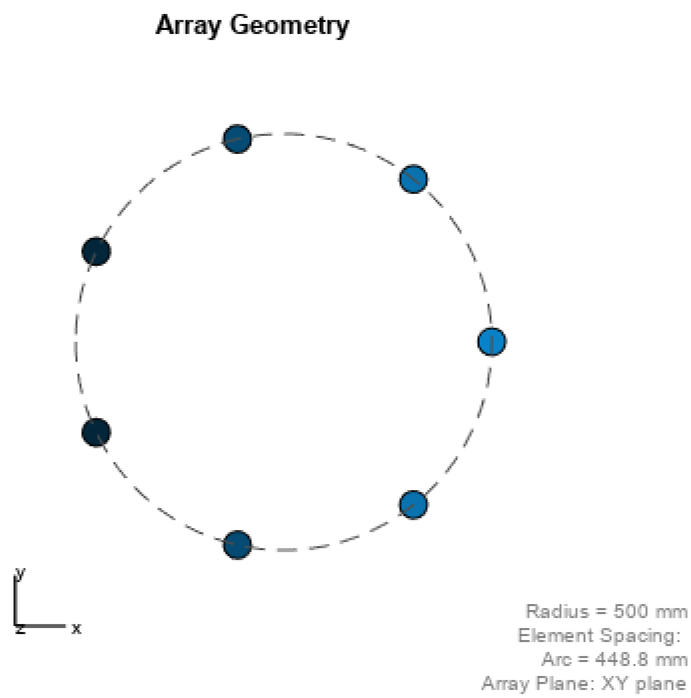
Construct a 7-element UCA array of isotropic antenna elements with a Taylor window taper. Design the array to have a radius of 0.5 meters. Then, draw the array showing the element taper shading.

```
Nelem = 7;
R = 0.5;
taper = taylorwin(Nelem);
sArray = phased.UCA(Nelem,R,'Taper',taper.');
w = getTaper(sArray)
```

```
w = 7×1

    0.4520
    0.9009
```

```
    1.3680
    1.5581
    1.3680
    0.9009
    0.4520
```

```
viewArray(sArray,'ShowTaper',true);
```



Both the output and figure above shows that the taper magnitudes are largest near the middle element.

Version History

Introduced in R2015a

isPolarizationCapable

System object: `phased.UCA`
Package: `phased`

Polarization capability

Syntax

```
flag = isPolarizationCapable(H)
```

Description

`flag = isPolarizationCapable(H)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization when all of its constituent sensor elements support polarization.

Input Arguments

`H` — Uniform line array
System object

Uniform line array specified as a `phased.UCA` System object.

Output Arguments

`flag` — Polarization-capability flag
boolean

Polarization-capability flag returned as a boolean value `true` when the array supports polarization or `false` when it does not.

Examples

Show UCA is Polarization Capable

Determine whether a UCA array of 7 short-dipole antenna elements supports polarization. The array radius is one-half meter.

```
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[1e9 10e9]);
array = phased.UCA('NumElements',7,'Radius',0.5,'Element',antenna);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

The returned value 1 from `isPolarizationCapable` shows that a UCA of short-dipole antenna elements supports polarization.

Version History

Introduced in R2015a

pattern

System object: phased.UCA
Package: phased

Plot UCA array pattern

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

pattern(sArray,FREQ) plots the 3-D array directivity pattern (in dBi) for the array specified in sArray. The operating frequency is specified in FREQ.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

pattern(sArray,FREQ,AZ) plots the array directivity pattern at the specified azimuth angle.

pattern(sArray,FREQ,AZ,EL) plots the array directivity pattern at specified azimuth and elevation angles.

pattern(___ ,Name,Value) plots the array pattern with additional options specified by one or more Name,Value pair arguments.

[PAT,AZ_ANG,EL_ANG] = pattern(___) returns the array pattern in PAT. The AZ_ANG output contains the coordinate values corresponding to the rows of PAT. The EL_ANG output contains the coordinate values corresponding to the columns of PAT. If the 'CoordinateSystem' parameter is set to 'uv', then AZ_ANG contains the *U* coordinates of the pattern and EL_ANG contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Input Arguments

sArray — Uniform circular array

System object

Uniform circular array, specified as a phased.UCA System object.

Example: sArray= phased.UCA;

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`AZ` — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

`EL` — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: 'uv'

Data Types: char

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Orientation — Array orientation
[0;0;0]. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

ShowArray — View array geometry
false (default) | true

View the array geometry along with the 3D radiation pattern, specified as false or true.

Data Types: logical

ShowLocalCoordinates — Show local coordinate axes
true (default) | false

Show the local coordinate axes, specified as true or false.

Data Types: logical

ShowColorbar — Show colorbar
true (default) | false

Show the colorbar, specified as true or false.

Data Types: logical

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.

Weights Dimension	FREQ Dimension	Purpose
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

PAT — Array pattern

M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of `PAT` correspond to the dimensions of the output arguments `AZ_ANG` and `EL_ANG`.

AZ_ANG — Azimuth angles

scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in `AZ`. The columns of `PAT` correspond to the values in `AZ_ANG`. Units are in degrees.

EL_ANG — Elevation angles

scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

Examples

Pattern of 11-Element UCA Antenna Array

Create an 11-element uniform circular array (UCA) having a 1.5 m radius and operating at 500 MHz. The array consists of short-dipole antenna elements. First, display the vertical component of the response at 45 degrees azimuth and 0 degrees elevation. Then plot the azimuth and elevation directivities.

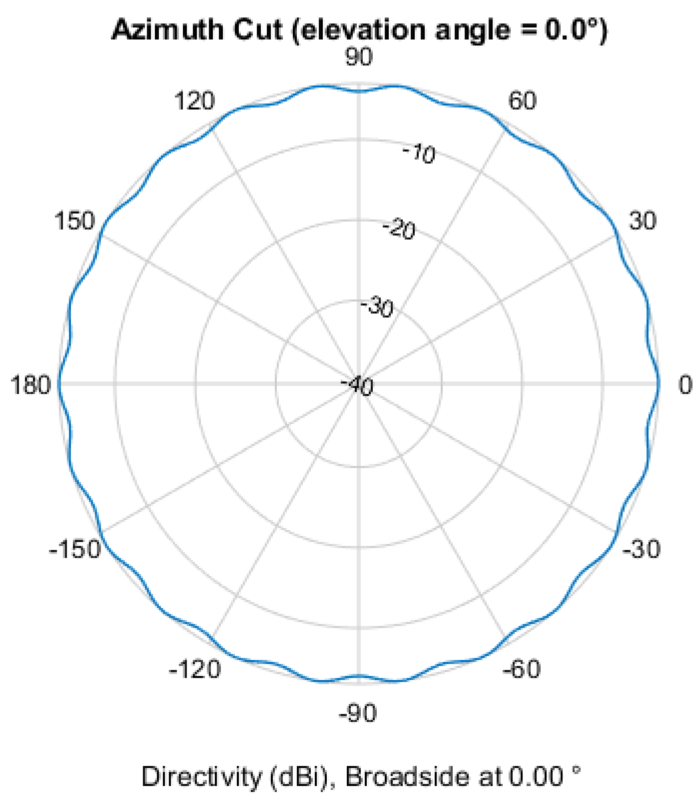
```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[50e6,1000e6],...
```

```
    'AxisDirection','Z');
array = phased.UCA('NumElements',11,'Radius',1.5,'Element',antenna);
fc = 500e6;
ang = [45;0];
resp = array(fc,ang);
disp(resp.V)
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

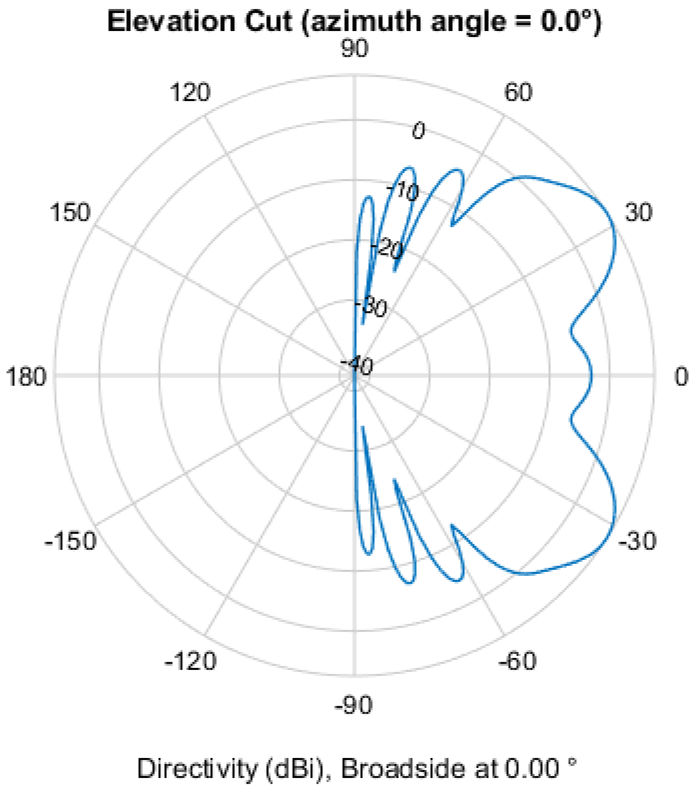
Display the azimuth directivity pattern at 500 MHz for azimuth angles between -180 and 180 degrees.

```
c = physconst('LightSpeed');
pattern(array,fc,[-180:180],0,'Type','directivity','PropagationSpeed',c)
```



Display the elevation directivity pattern at 500 MHz for elevation angles between -90 and 90 degrees.

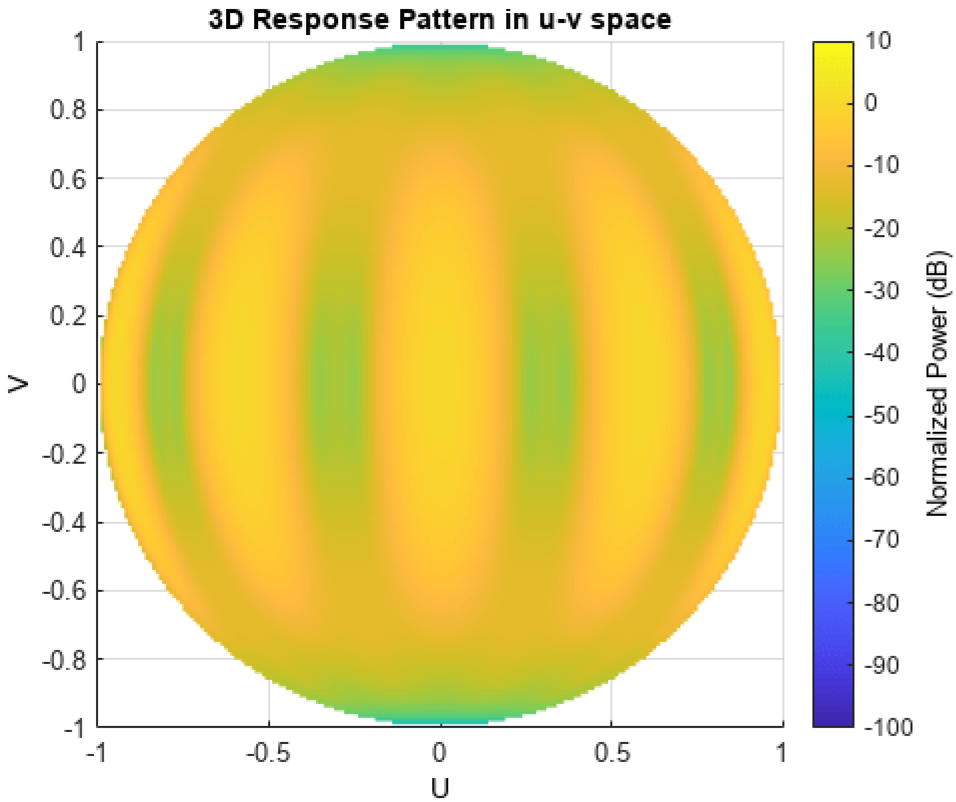
```
pattern(array,fc,[0],[-90:90],'Type','directivity','PropagationSpeed',c)
```



Pattern of 10-Element UCA Antenna Array in UV Space

Create a 10-element UCA antenna array consisting of cosine antenna elements. Display the 3-D power pattern in UV space.

```
sCos = phased.CosineAntennaElement('FrequencyRange',[100e6 1e9],...
    'CosinePower',[2.5,2.5]);
sUCA = phased.UCA('NumElements',10,...
    'Radius',1.5,...
    'Element',sCos);
c = physconst('LightSpeed');
fc = 500e6;
pattern(sUCA,fc,[-1:.01:1],[-1:.01:1],...
    'CoordinateSystem','uv',...
    'Type','powerdb',...
    'PropagationSpeed',c)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\text{rad}}(\theta,\varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also
`patternAzimuth` | `patternElevation`

patternAzimuth

System object: phased.UCA
Package: phased

Plot UCA array directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

patternAzimuth(sArray,FREQ) plots the 2-D array directivity pattern versus azimuth (in dBi) for the array sArray at zero degrees elevation angle. The argument FREQ specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

patternAzimuth(sArray,FREQ,EL), in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array sArray at the elevation angle specified by EL. When EL is a vector, multiple overlaid plots are created.

patternAzimuth(sArray,FREQ,EL,Name,Value) plots the array pattern with additional options specified by one or more Name,Value pair arguments.

PAT = patternAzimuth(___) returns the array pattern. PAT is a matrix whose entries represent the pattern at corresponding sampling points specified by the 'Azimuth' parameter and the EL input argument.

Input Arguments

sArray — Uniform circular array

System object

Uniform circular array, specified as a phased.UCA System object.

Example: sArray= phased.UCA;

FREQ — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

`PAT` — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

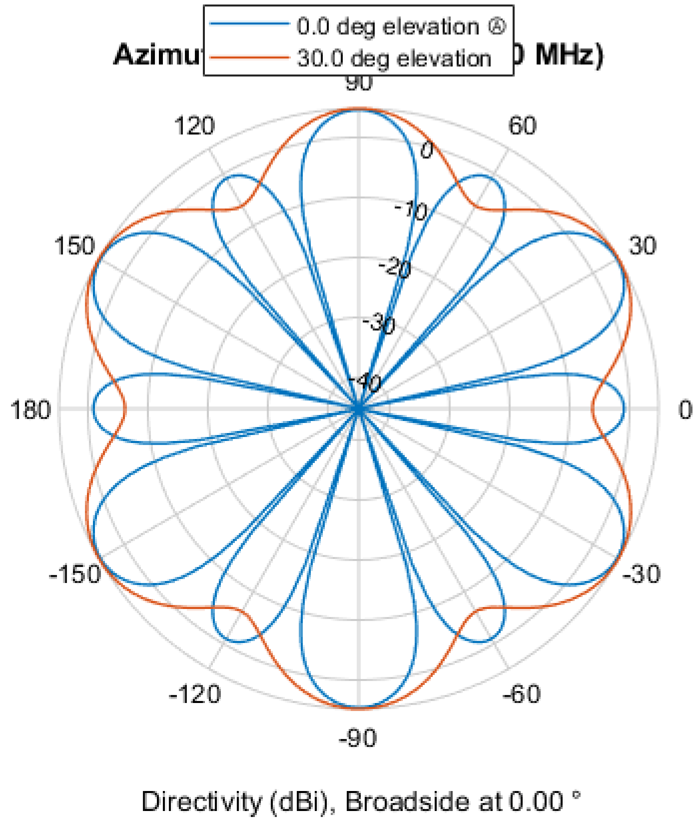
Examples

Plot Azimuth Pattern of UCA

Create a 6-element UCA of short-dipole antenna elements. Design the array to have a radius of 0.5 meters. Plot an azimuth cut of directivity at 0 and 10 degrees elevation. Assume the operating frequency is 500 MHz.

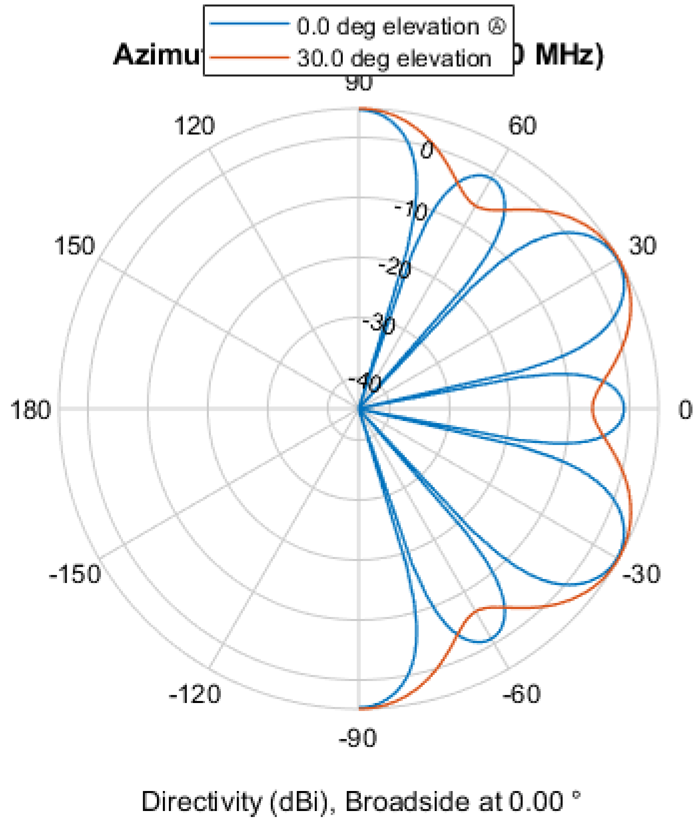
```
fc = 500e6;
sCDant = phased.ShortDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
```

```
sUCA = phased.UCA('NumElements',6,'Radius',0.5,'Element',sCDant);
patternAzimuth(sUCA,fc,[0 30])
```



You can plot a smaller range of azimuth angles by setting the `Azimuth` property.

```
patternAzimuth(sUCA,fc,[0 30],'Azimuth',[-90:90])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: `phased.UCA`
Package: `phased`

Plot UCA array directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sArray,FREQ)` plots the 2-D array directivity pattern versus elevation (in dBi) for the array `sArray` at zero degrees azimuth angle. When `AZ` is a vector, multiple overlaid plots are created. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternElevation(sArray,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sArray,FREQ,AZ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sArray` — Uniform circular array

System object

Uniform circular array, specified as a `phased.UCA` System object.

Example: `sArray= phased.UCA;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

`AZ` — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of 'Weights' and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the phased.SteeringVector System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as phased.Radiator or phased.Collector. However, for the directivity, pattern, patternAzimuth, and patternElevation methods of any array System object use the steering vector without conjugation.

Example: 'Weights',ones(10,1)

Data Types: double
Complex Number Support: Yes

Elevation — Elevation angles
[-90:90] (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of 'Elevation' and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: 'Elevation',[-90:2:90]

Data Types: double

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the 'Elevation' name-value pair argument. The dimension *N* is the number of azimuth angles determined by the AZ argument.

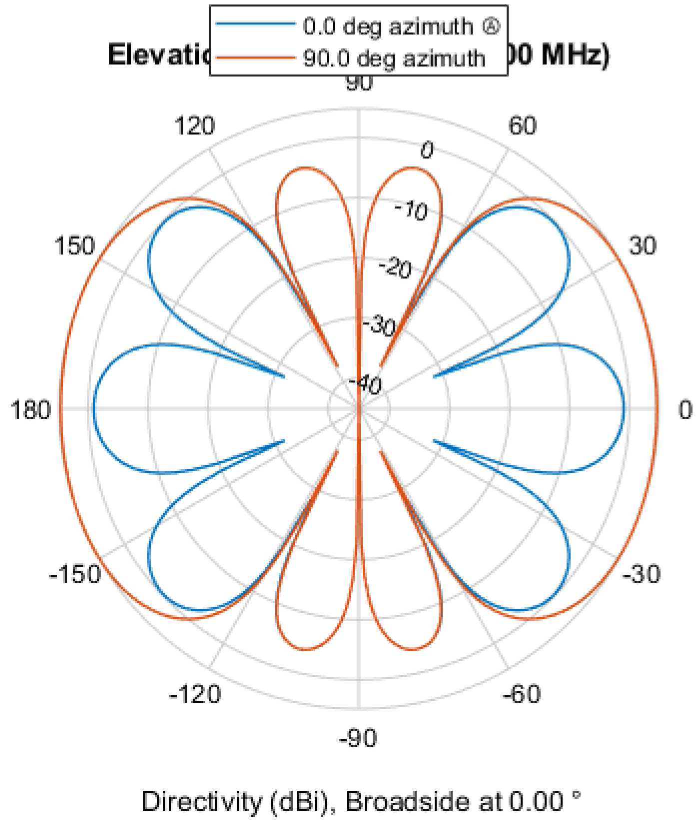
Examples

Plot Elevation Pattern of UCA

Create a 6-element UCA of short-dipole antenna elements. Design the array to have a radius of 0.5 meters. Plot an elevation cut of directivity at 0 and 90 degrees azimuth. Assume the operating frequency is 500 MHz.

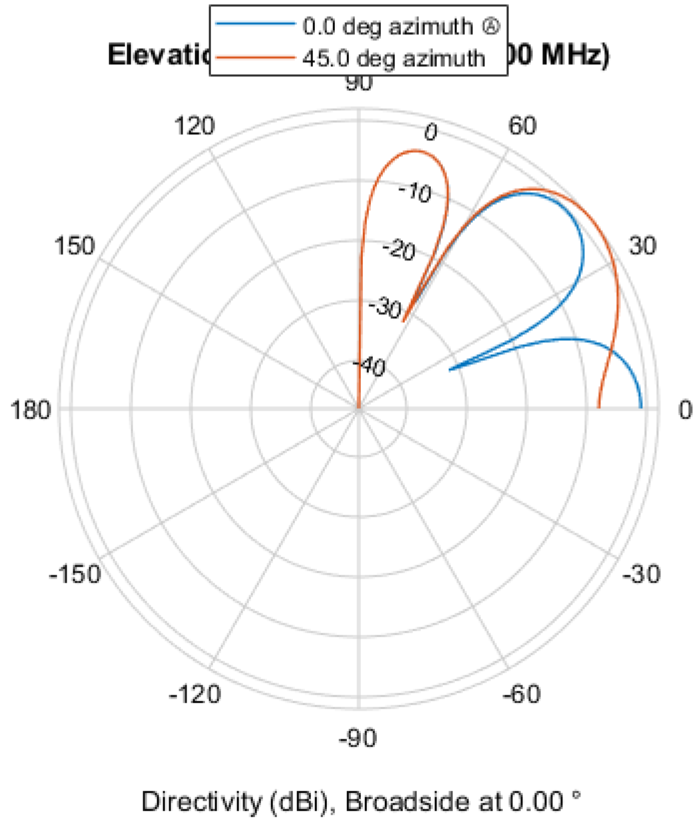
```
fc = 500e6;
sCDant = phased.ShortDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
```

```
sUCA = phased.UCA('NumElements',6,'Radius',0.5,'Element',sCDant);
patternElevation(sUCA,fc,[0 90])
```



You can plot a smaller range of elevation angles by setting the `Elevation` property.

```
patternElevation(sUCA,fc,[0 45],'Elevation',[0:90])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth`

step

System object: `phased.UCA`
Package: `phased`

Output responses of array elements

Syntax

`RESP = step(sArray,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(sArray,FREQ,ANG)` returns the responses, `RESP`, of the array elements, at operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`sArray` — Uniform circular array
System object

Uniform circular array, specified as a `phased.UCA` System object.

Example: `sArray= phased.UCA;`

`FREQ` — Operating frequency
positive scalar | 1-by-*L* real-valued row vector

Operating frequency of array specified, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For antenna or microphone elements, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the array response is returned as zero. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as zero.

Example: `[1e8 2e8]`

Data Types: `double`

ANG — Response directions
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Response directions, specified as either a 2-by-*M* real-valued matrix or a real-valued row vector of length *M*.

If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive. Angle units are in degrees.

If `ANG` is a row vector of length *M*, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be `0°`.

Example: `[20;15]`

Data Types: `double`

Output Arguments

RESP — Voltage responses of phased array
complex-valued *N*-by-*M*-by-*L* matrix | complex-valued structure

Voltage responses of a phased array, specified as a complex-valued matrix or a `struct` with complex-valued fields. The output depends on whether the array supports polarization or not.

- If the array elements do not support polarization, the voltage response, `RESP`, has the dimensions *N*-by-*M*-by-*L*.
  - *N* (rows) is the number of elements in the array
  - *M* (columns) is the number of angles specified in `ANG`
  - *L* (pages) is the number of frequencies specified in `FREQ`

  For each array element, the columns of `RESP` contain the array element responses for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP` contains the array element responses for the corresponding frequency specified in `FREQ`.
- If the array supports polarization, `RESP` is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*.
  - *N* (rows) is the number of elements in the array
  - *M* (columns) is the number of angles specified in `ANG`
  - *L* (pages) is the number of frequencies specified in `FREQ`

  For each array element, the columns of `RESP.H` or `RESP.V` contain the array element responses for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP.H` or `RESP.V` contains the array element responses for the corresponding frequency specified in `FREQ`.

Examples

Response of UCA Array

Create a 5-element uniform circular array (UCA) of cosine antenna elements having a 0.5 meter radius. Find the element responses at the 0 degrees azimuth and elevation at a 300 MHz operating frequency.

```
c = physconst('LightSpeed');
fc = 300e6;
sCos = phased.CosineAntennaElement('CosinePower',[1,1]);
sArray = phased.UCA('Element',sCos,'NumElements',5,'Radius',0.5);
ang = [0;0];
resp = step(sArray,fc,ang)
```

```
resp = 5×1

         0
    0.3090
    1.0000
    0.3090
         0
```

Version History

Introduced in R2015a

viewArray

System object: `phased.UCA`
Package: `phased`

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

`viewArray(H)` plots the geometry of the array specified in `H`.

`viewArray(H,Name,Value)` plots the geometry of the array, with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = viewArray( ___ )` returns the handle of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

`H` — Uniform circular array
System object

Uniform circular array specified as a `phased.UCA` System object.

Example: `phased.UCA()`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Parent` — Handle to axes
real-valued scalar

Handle to the axes along which the array geometry is displayed.

`ShowNormals` — Option to show normal vectors
`false` (default) | `true`

Option to show normal directions, specified as the comma-separated pair consisting of `'ShowNormals'` and a Boolean value.

- `true` — show the normal directions of all elements in the array
- `false` — plot the elements without showing normal directions

Example: `false`

Data Types: `logical`

`ShowLocalCoordinates` — Show local coordinates
`true` (default) | `false`

Logical flag specifying whether to show the local coordinate axes.

Data Types: `logical`

`ShowAnnotation` — Show aperture size and element spacing annotations
`true` (default) | `false`

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Data Types: `logical`

`Orientation` — Array orientation
`[0;0;0]` (default) | 3-by-1 real-valued vector

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

Data Types: `double`

`ShowTaper` — Option to show taper magnitude
`false` (default) | `true`

Option to show taper magnitude, specified as the comma-separated pair consisting of `'ShowTaper'` and a Boolean value.

- `true` — change the element color brightness in proportion to the element taper magnitude
- `false` — plot all elements using the same color

Data Types: `logical`

`ShowIndex` — Element indices to show
`'None'` (default) | vector of positive integers | `'All'`

Element indices to show in the figure, specified as the comma-separated pair consisting of `'ShowIndex'` and a vector of positive integers. Each number in the vector must be an integer between 1 and the number of elements. To show all of indices of the array, specify `'All'`. To suppress all indices, specify `'None'`.

Example: `[1,2,3]`

Data Types: `double`

`Title` — Plot title
`'Array Geometry'` (default) | character vector

Plot title, specified as a character vector.

Example: 'My array plot'

Output Arguments

`hPlot` — Handle of array elements
scalar

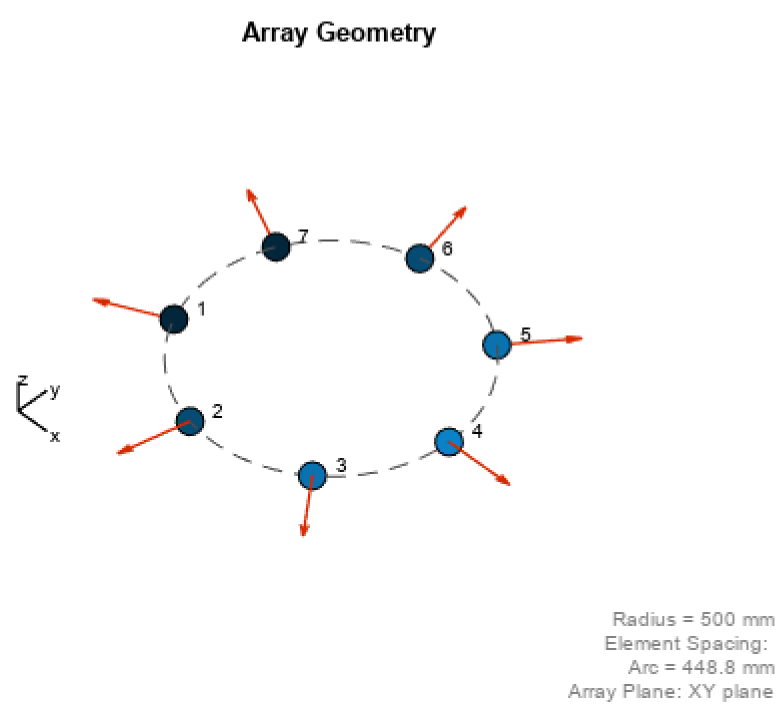
Handle of array elements in the figure window, specified as a scalar.

Examples

View UCA Array

Construct an 7-element UCA of isotropic antenna elements with a Taylor window taper. Design the array to have a radius of 0.5 meters. Then, draw the array showing the element normals, element indices, and element taper shading.

```
Nelem = 7;
R = 0.5;
taper = taylorwin(Nelem);
sArray = phased.UCA(Nelem,R,'Taper',taper.');
w = getTaper(sArray);
viewArray(sArray,'ShowNormals',true,'ShowIndex','All','ShowTaper',true);
```



Version History

Introduced in R2015a

See Also

`phased.ArrayResponse`

Topics

Phased Array Gallery

phased.ULA

Package: phased

Uniform linear array

Description

phased.ULA creates a uniform linear array (ULA) System object and computes its response.

To compute the response for each element in the array for specified directions:

1. Create the phased.ULA object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
array = phased.ULA
array = phased.ULA(Name=Value)
array = phased.ULA(N,D,Name=Value)
```

Description

array = phased.ULA creates a uniform linear array (ULA) System object. In this syntax, the object models a ULA formed with identical isotropic phased array sensor elements. The origin of the local coordinate system is the phase center of the array. The positive *x*-axis is the direction normal to the array, and the elements of the array are located along the *y*-axis.

array = phased.ULA(Name=Value) creates the object array with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1 = Value1, ... ,NameN = ValueN).

array = phased.ULA(N,D,Name=Value) creates a ULA array object with the NumElements property set to N, the ElementSpacing property set to D, and other specified property Names set to the specified Values. N and D are value-only arguments. When specifying a value-only argument, specify all preceding value-only arguments. You can specify name-value pair arguments in any order.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Element` — phased array element
isotropic antenna element System object with default properties (default) | Phased Array System Toolbox antenna, microphone, or transducer element System object | Antenna Toolbox antenna System object

Phased array element, specified as a Phased Array System Toolbox antenna, microphone, or transducer element or Antenna Toolbox antenna.

Example: `phased.CosineAntennaElement`

`NumElements` — Number of array elements
2 (default) | positive integer

Number of array elements, specified as a positive integer.

Data Types: `double`

`ElementSpacing` — Array element spacing
`0.5` (default) | positive scalar

Spacing between two adjacent elements in the array, specified as a positive scalar. Units are in meters.

Data Types: `double`

`ArrayAxis` — Axis of linear array
`'y'` (default) | `'x''z'`

Axis of linear array, specified as `'y'`, `'x'`, or `'z'`. Linear array elements are located along the selected coordinate system axis. The array axis determines the direction along which the element normal vectors point.

ArrayAxis Property Value	Element Normal Direction
`'x'`	azimuth = 90°, elevation = 0° (*y*-axis)
`'y'` (default)	azimuth = 0°, elevation = 0° (*x*-axis)
`'z'`	azimuth = 0°, elevation = 0° (*x*-axis)

Example: `'z'`

Data Types: `char` | `string`

`Taper` — Array element tapers
1 (default) | complex-valued scalar | 1-by-*N* complex-valued row vector | *N*-by-1 c column vector

Array element tapering, specified as a complex-valued scalar, 1-by-*N* complex-valued row vector, or *N*-by-1 complex-valued column vector. *N* represents the number of elements of the array. Tapers, also known as weights, are applied to each sensor element in the sensor array and modify both the amplitude and phase of the received data. If `'Taper'` is a scalar, the same taper value is applied to all elements. If `'Taper'` is a vector, each taper value is applied to the corresponding sensor element.

Data Types: `double`

Usage

Syntax

```
resp = array(freq,ang)
```

Description

`resp = array(freq,ang)` returns the array element responses, `resp`, at the operating frequencies specified in `freq` and in directions specified in `ang`.

Input Arguments

`freq` — Array operating frequencies
length-*L* row vector of positive values

Array operating frequencies, specified as a length-*L* row vector. Typical values are within the range specified by a frequency range property of the array element. That property is called `FrequencyRange` or `FrequencyVector`, depending on the type of array element. The element has zero response at frequencies outside that range. Units are in Hz.

Data Types: `double`

`ang` — Array response directions
length-*M* row vector with real values | 2-by-*M* matrix with real values

Array response directions, specified as a 2-by-*M* matrix or length-*M* row vector.

If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

If `ang` is a row vector of length *M*, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be 0°.

Units are in degrees.

Output Arguments

`resp` — Voltage responses of phased array
N-by-*M*-by-*L* complex-valued MATLAB array | MATLAB struct

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, `resp`, has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array. The dimension *M* is the number of angles specified in `ang`. *L* is the number of frequencies specified in `freq`. For any element, the columns of `resp` contain the responses of the array elements for the corresponding direction specified in `ang`. Each of the *L* pages of `resp` contains the responses of the array elements for the corresponding frequency specified in `freq`.
- If the array is capable of supporting polarization, the voltage response, `resp`, is a MATLAB `struct` containing two fields, `resp.H` and `resp.V`. The field, `resp.H`, represents the array's horizontal polarization response, while `resp.V` represents the array's vertical polarization

response. Each field has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array, and *M* is the number of angles specified in `ang`. *L* is the number of frequencies specified in `freq`. Each column of `resp` contains the responses of the array elements for the corresponding direction specified in `ang`. Each of the *L* pages of `ang` contains the responses of the array elements for the corresponding frequency specified in `freq`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to phased.ULA and other array System objects

beamwidth	Compute and display beamwidth of an array
collectPlaneWave	Simulate received plane waves at array
directivity	Compute array directivity
getElementNormal	Normal vectors for array elements
getElementPosition	Positions of array elements
getNumElements	Number of elements in an array
getTaper	Array element tapers
isPolarizationCapable	Array polarization capability
pattern	Plot array directivity and patterns
patternAzimuth	Plot array directivity or pattern versus azimuth
patternElevation	Plot array directivity or pattern versus elevation
perturbations	Perturbations defined on array
perturbedArray	Apply perturbations to phased array
perturbedPattern	Display pattern of perturbed array
plotGratingLobeDiagram	Plot grating lobe diagram of array
viewArray	View array geometry

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Directivity of Uniform Linear Array

Compute the directivities of two different uniform linear arrays (ULA). One array consists of isotropic antenna elements and the second array consists of cosine antenna elements. In addition, compute the directivity when the first array is steered in a specified direction. For each case, calculate the directivities for a set of seven different azimuth directions all at zero degrees elevation. Set the frequency to 800 MHz.

Array of isotropic antenna elements

First, create a 10-element ULA of isotropic antenna elements spaced 1/2-wavelength apart.

```
c = physconst('LightSpeed');
fc = 3e8;
lambda = c/fc;
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
myAnt1 = phased.IsotropicAntennaElement;
myArray1 = phased.ULA(10,lambda/2,'Element',myAnt1);
```

Compute the directivity.

```
d = directivity(myArray1,fc,ang,'PropagationSpeed',c)
```

```
d = 7×1

   -6.9886
   -6.2283
   -6.5176
   10.0011
   -6.5176
   -6.2283
   -6.9886
```

Array of cosine antenna elements

Next, create a 10-element ULA of cosine antenna elements spaced 1/2-wavelength apart.

```
myAnt2 = phased.CosineAntennaElement('CosinePower',[1.8,1.8]);
myArray2 = phased.ULA(10,lambda/2,'Element',myAnt2);
```

Compute the directivity.

```
d = directivity(myArray2,fc,ang,'PropagationSpeed',c)
```

```
d = 7×1

   -1.9838
    0.0529
    0.4968
   17.2548
    0.4968
    0.0529
   -1.9838
```

The directivity of the cosine ULA is greater than the directivity of the isotropic ULA because of the larger directivity of the cosine antenna element.

Steered array of isotropic antenna elements

Finally, steer the isotropic antenna array to 30 degrees in azimuth and compute the directivity.

```
w = steervec(getElementPosition(myArray1)/lambda,[30;0]);
d = directivity(myArray1,fc,ang,'PropagationSpeed',c,...
    'Weights',w)
```

```
d = 7×1

 -297.2705
  -13.9783
```

```
   -9.5713
   -6.9897
   -4.5787
   -2.0536
   10.0000
```

The directivity is greatest in the steered direction.

ULA Element Normals

Construct three ULAs with elements along the *x*-, *y*-, and *z*-axes. Obtain the element normals.

First, choose the array axis along the *x*-axis.

```
sULA1 = phased.ULA('NumElements',5,'ArrayAxis','x');
norm = getElementNormal(sULA1)
```

```
norm = 2×5

    90    90    90    90    90
     0     0     0     0     0
```

The element normal vectors point along the *y*-axis.

Next, choose the array axis along the *y*-axis.

```
sULA2 = phased.ULA('NumElements',5,'ArrayAxis','y');
norm = getElementNormal(sULA2)
```

```
norm = 2×5

     0     0     0     0     0
     0     0     0     0     0
```

The element normal vectors point along the *x*-axis.

Finally, set the array axis along the *z*-axis. Obtain the normal vectors of the odd-numbered elements.

```
sULA3 = phased.ULA('NumElements',5,'ArrayAxis','z');
norm = getElementNormal(sULA3,[1,3,5])
```

```
norm = 2×3

     0     0     0
     0     0     0
```

The element normal vectors also point along the *x*-axis.

ULA Element Positions

Construct a ULA with 5 elements along the z-axis. Obtain the element positions.

```
sULA = phased.ULA('NumElements',5,'ArrayAxis','z');
pos = getElementPosition(sULA)
```

```
pos = 3×5

         0         0         0         0         0
         0         0         0         0         0
   -1.0000   -0.5000         0    0.5000    1.0000
```

Get Number of ULA Elements

Construct a default ULA and obtain the number of elements in that array.

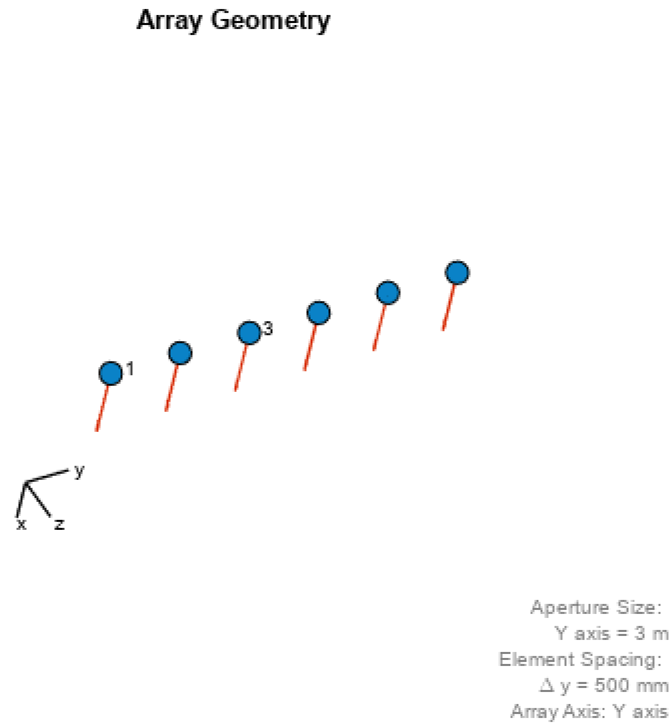
```
 array = phased.ULA;
 N = getNumElements(array)
```

```
N = 2
```

Geometry and Indices of ULA Elements

Draw a 6-element ULA and use the 'ShowIndex' parameter to show the indices of the first and third elements.

```
array = phased.ULA(6);
viewArray(array,'ShowIndex',[1 3],'ShowNormals',true, ...
    'ShowLocalCoordinates',true,'Orientation',[60;100;45], ...
    'ShowAnnotation',true)
```



Construct ULA with Taylor Window

Construct a 5-element ULA with a Taylor window taper. Then, obtain the element taper values.

```
taper = taylorwin(5)';
array = phased.ULA(5,'Taper',taper);
w = getTaper(array)
```

```
w = 5×1

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

Short-Dipole Antenna ULA Supports Polarization

Show that an array of `phased.ShortDipoleAntennaElement` antenna elements supports polarization.

```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[1e9 10e9]);
array = phased.ULA('NumElements',3,'Element',antenna);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

The returned value of `1` shows that this array supports polarization.

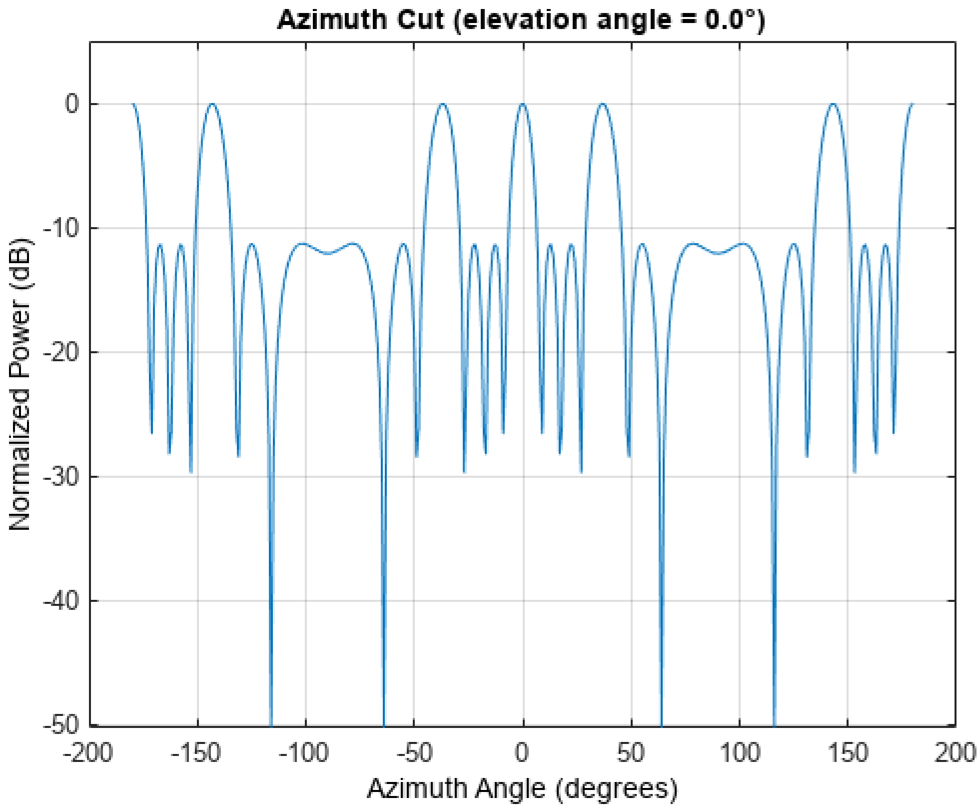
Plot Pattern of 4-Element Antenna Array

Create a 4-element undersampled ULA and find the response of each element at boresight. Plot the array pattern at 1 GHz for azimuth angles between -180 and 180 degrees. The default element spacing is 0.5 meters.

```
array = phased.ULA('NumElements',4);
fc = 1e9;
ang = [0;0];
resp = array(fc,ang)
```

```
resp = 4×1

     1
     1
     1
     1
```

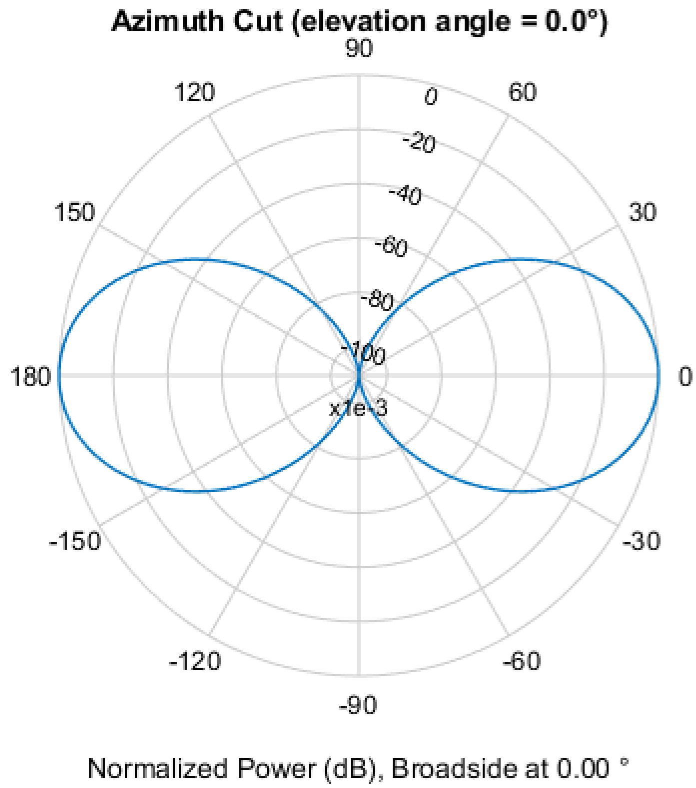
```
c = physconst('LightSpeed');
pattern(array,fc,-180:180,0,'PropagationSpeed',c,...
    'CoordinateSystem','rectangular',...
    'Type','powerdb','Normalize',true)
```



Plot Pattern of 10-Element Microphone ULA

Construct a 10-element uniform linear array of omnidirectional microphones spaced 3 cm apart. Then, plot the array pattern at 100 Hz.

```
mic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
Nele = 10;
array = phased.ULA('NumElements',Nele,...
    'ElementSpacing',3e-2,...
    'Element',mic);
fc = 100;
ang = [0; 0];
resp = array(fc,ang);
c = 340;
pattern(array,fc,[-180:180],0,'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Type','powerdb',...
    'Normalize',true);
```



Plot Pattern of Array of Polarized Short-Dipole Antennas

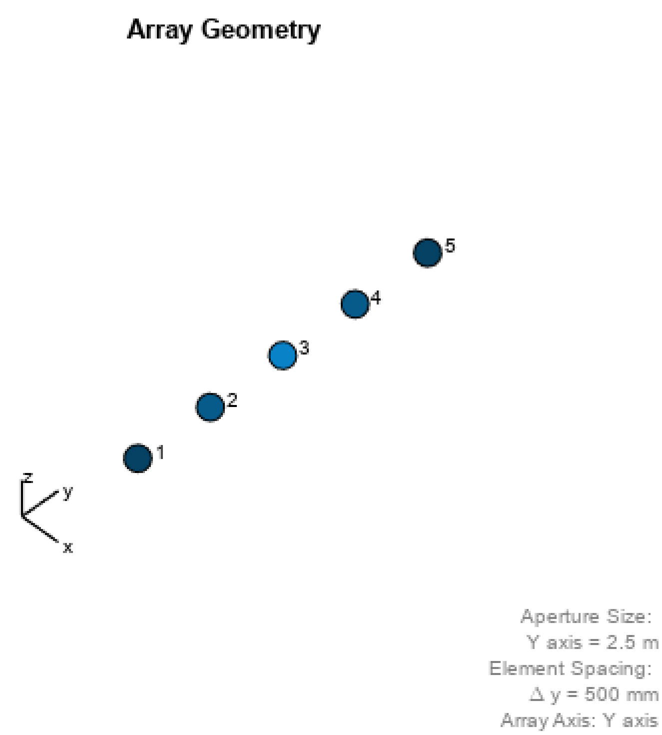
Build a tapered uniform line array of 5 short-dipole sensor elements. Because short dipoles support polarization, the array should as well. Verify that it supports polarization by looking at the output of the `isPolarizationCapable` method.

```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],'AxisDirection','Z');
array = phased.ULA('NumElements',5,'Element',antenna,...
    'Taper',[.5,.7,1,.7,.5]);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

Then, draw the array using the `viewArray` method.

```
viewArray(array,'ShowTaper',true,'ShowIndex','All')
```



Compute the horizontal and vertical responses.

```
fc = 150e6;
ang = [10];
resp = array(fc,ang);
```

Display the horizontal polarization response.

```
resp.H
```

```
ans = 5×1

     0
     0
     0
     0
     0
```

Display the vertical polarization response.

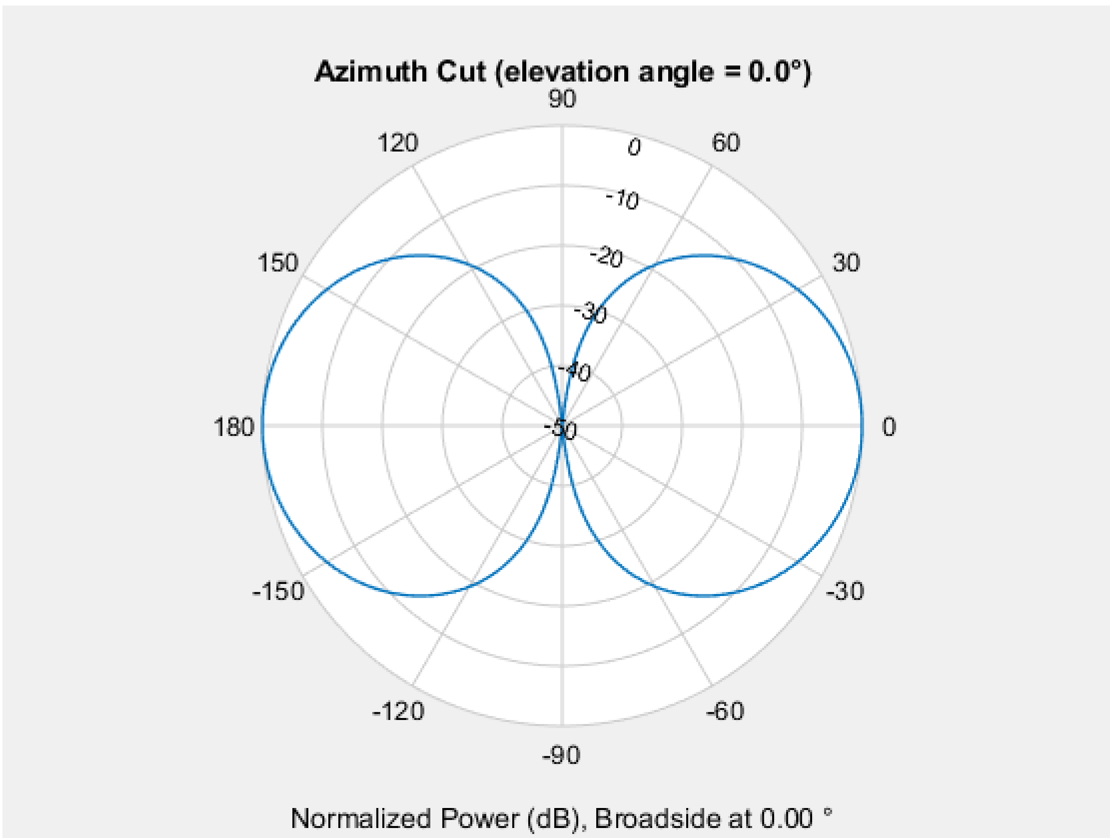
```
resp.V
```

```
ans = 5×1

   -0.6124
   -0.8573
   -1.2247
```

```
-0.8573
-0.6124
```

Plot an azimuth cut of the vertical polarization response.

```
c = physconst('LightSpeed');
pattern(array,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','polar',...
    'Polarization','V',...
    'Type','powerdb',...
    'Normalize',true)
```



Plot Pattern of 9-Element ULA Antenna Array of Short Dipoles

Create an 9-element ULA of short dipole antenna elements spaced 0.2 meters apart. Display the azimuth and elevation directivities. The operating frequency is 500 MHz. Plot the directivities in polar coordinates.

Evaluate the fields at 45 degrees azimuth and 0 degrees elevation.

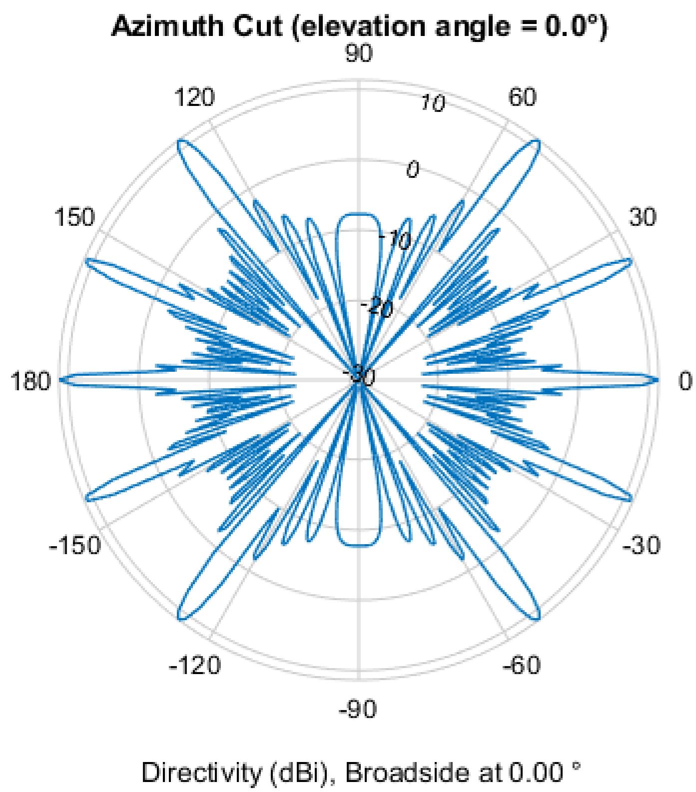
```
element = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[50e6,1000e6],...
    'AxisDirection','Z');
```

```
array = phased.ULA('NumElements',9,'ElementSpacing',1.5,'Element',element);
fc = 500e6;
ang = [45;0];
resp = array(fc,ang);
disp(resp.V)
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

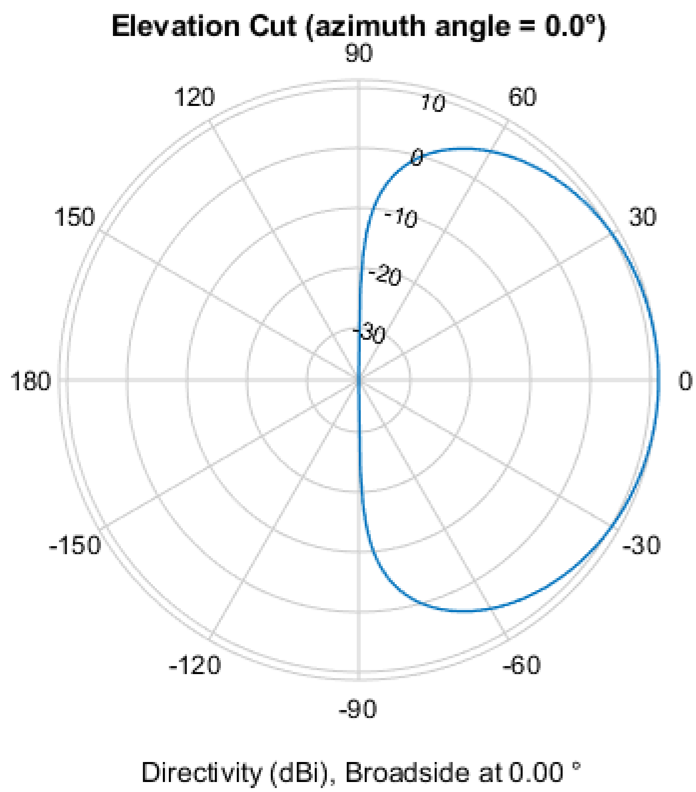
Display the azimuth directivity pattern at 500 MHz for azimuth angles between -180 and 180 degrees.

```
c = physconst('LightSpeed');
pattern(array,fc,[-180:180],0,...
    'Type','directivity',...
    'PropagationSpeed',c)
```



Display the elevation directivity pattern at 500 MHz for elevation angles between -90 and 90 degrees.

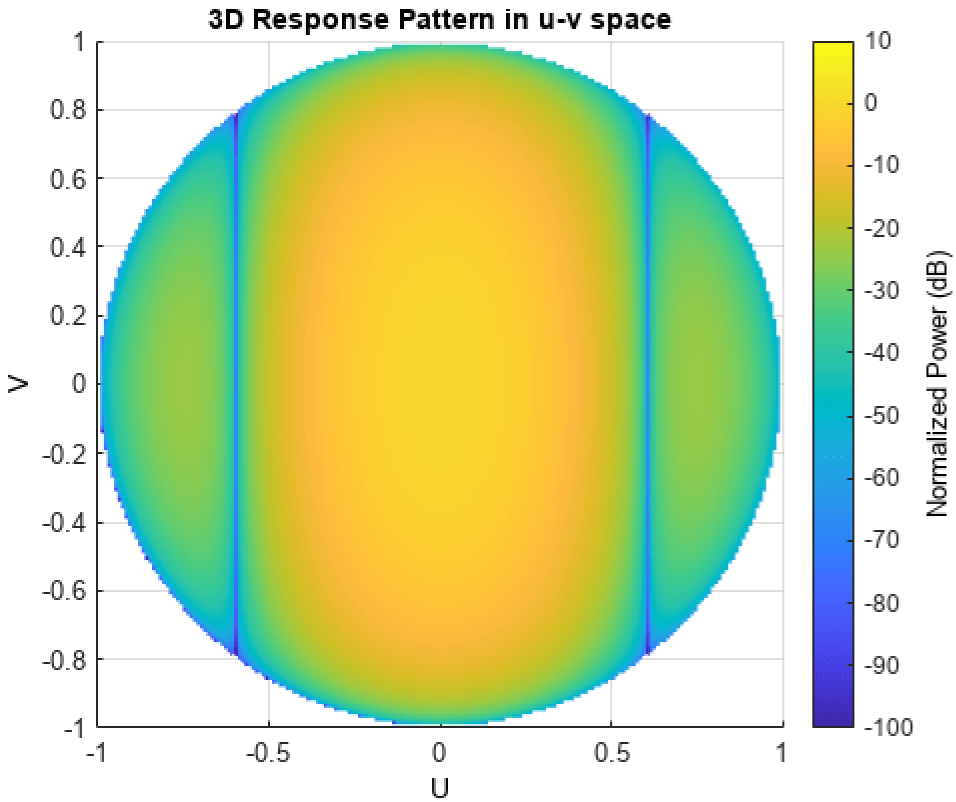
```
pattern(array,fc,[0],[-90:90],...
    'Type','directivity',...
    'PropagationSpeed',c)
```



Plot Pattern of 10-Element ULA Antenna Array in UV Space

Create a 10-element ULA antenna array consisting of cosine antenna elements spaced 10 cm apart. Display the 3-D power pattern in UV space. The operating frequency is 500 MHz.

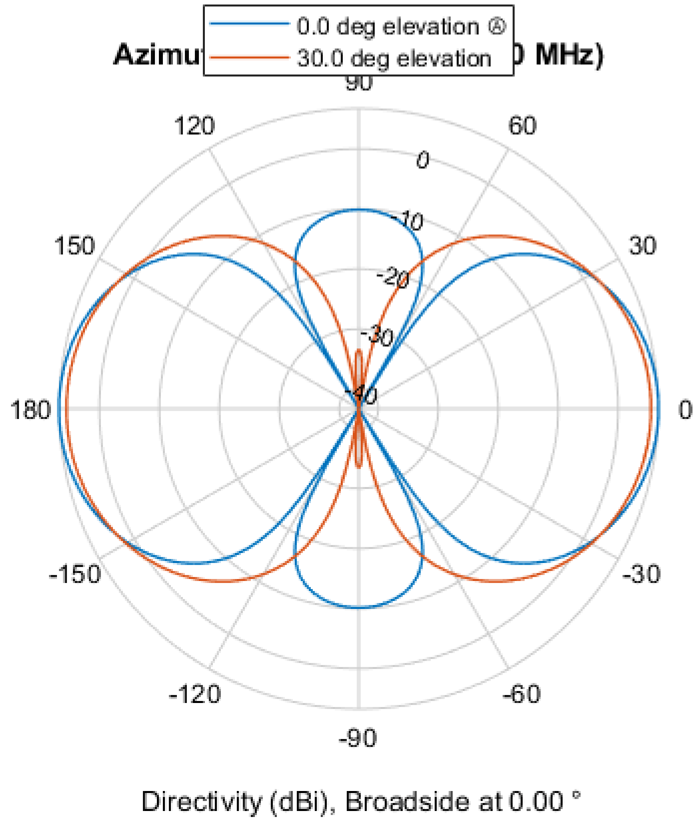
```
sCos = phased.CosineAntennaElement('FrequencyRange',[100e6 1e9],...
    'CosinePower',[2.5,2.5]);
sULA = phased.ULA('NumElements',10,...
    'ElementSpacing',.1,...
    'Element',sCos);
c = physconst('LightSpeed');
fc = 500e6;
pattern(sULA,fc,[-1:.01:1],[-1:.01:1],...
    'CoordinateSystem','uv',...
    'Type','powerdb',...
    'PropagationSpeed',c)
```



Plot Azimuth Pattern of ULA

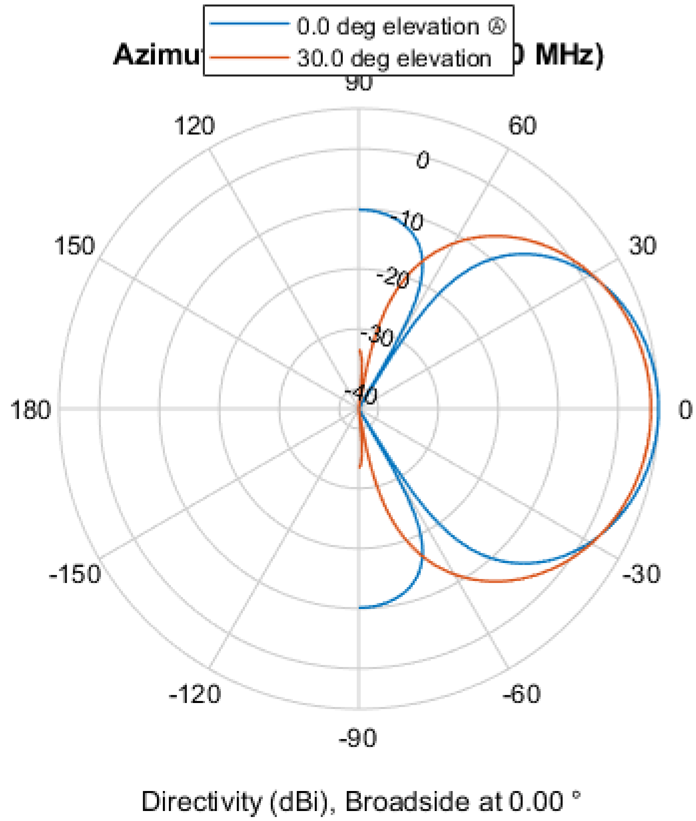
Create a 7-element ULA of short-dipole antenna elements spaced 10 cm apart. Plot an azimuth cut of directivity at 0 and 10 degrees elevation. Assume the operating frequency is 500 MHz.

```
fc = 500e6;
sCDant = phased.ShortDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
sULA = phased.ULA('NumElements',7,'ElementSpacing',0.1,'Element',sCDant);
patternAzimuth(sULA,fc,[0 30])
```



You can plot a smaller range of azimuth angles by setting the `Azimuth` property.

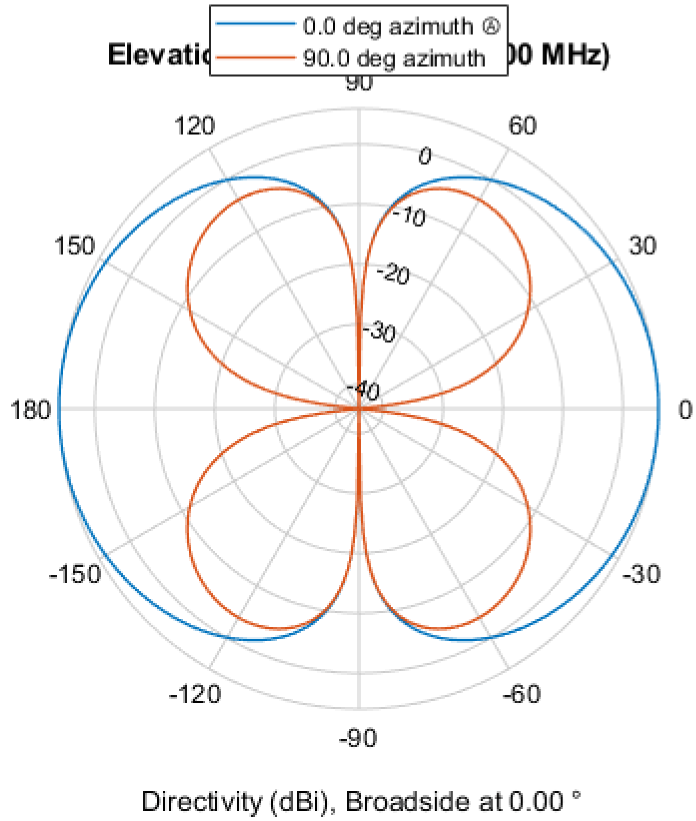
```
patternAzimuth(sULA,fc,[0 30],'Azimuth',[-90:90])
```



Plot Elevation Pattern of ULA

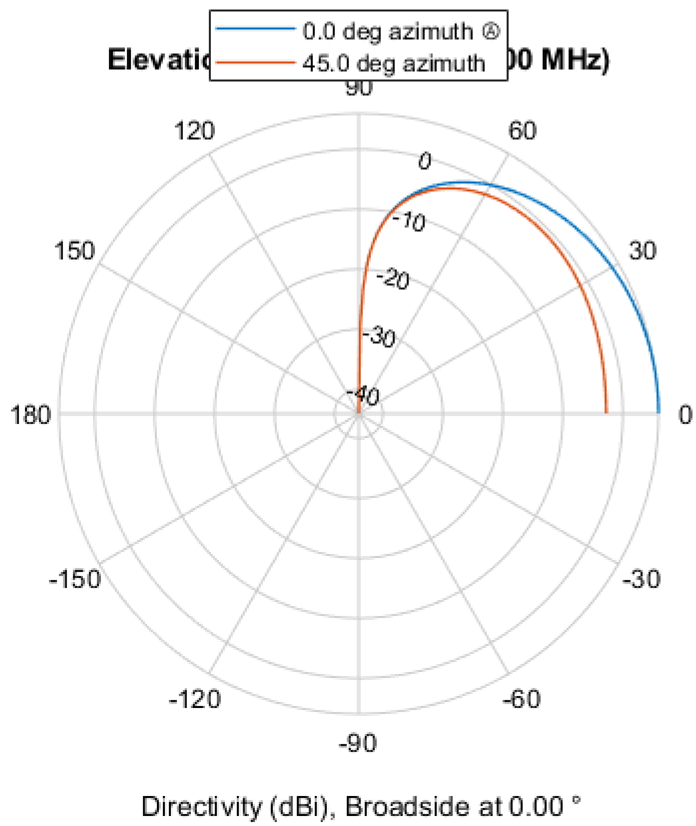
Create a 6-element ULA of short-dipole antenna elements with element spacing of 10 cm. Plot an elevation cut of directivity at 0 and 90 degrees azimuth. Assume the operating frequency is 500 MHz.

```
fc = 500e6;
c = physconst('LightSpeed');
sSD = phased.ShortDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
sULA = phased.ULA('NumElements',6,'ElementSpacing',0.1,'Element',sSD);
patternElevation(sULA,fc,[0 90],'PropagationSpeed',c)
```



You can plot a smaller range of elevation angles by setting the `Elevation` property.

```
patternElevation(sULA,fc,[0 45],'Elevation',[0:90],'PropagationSpeed',c)
```



Plot Beamwidth of Sonar Array

Plot the beamwidth of a sonar array operating at a frequency of 2 kHz when the propagation speed of sound in water is 1500 m/s.

The sonar array consists of a 20-element uniform linear array (ULA). Consider the element of the ULA to be a backbaffled `phased.IsotropicProjector` with a `VoltageResponse` of 100 Volts and with a `FrequencyRange` from 10 Hz to 300 kHz. Create a `phased.ULA` object to model the uniform linear array.

```
projector = phased.IsotropicProjector('BackBaffled',true,...
       'VoltageResponse',100,'FrequencyRange',[10 300000])
```

```
projector = 
  phased.IsotropicProjector with properties:

    VoltageResponse: 100
     FrequencyRange: [10 300000]
        BackBaffled: true
```

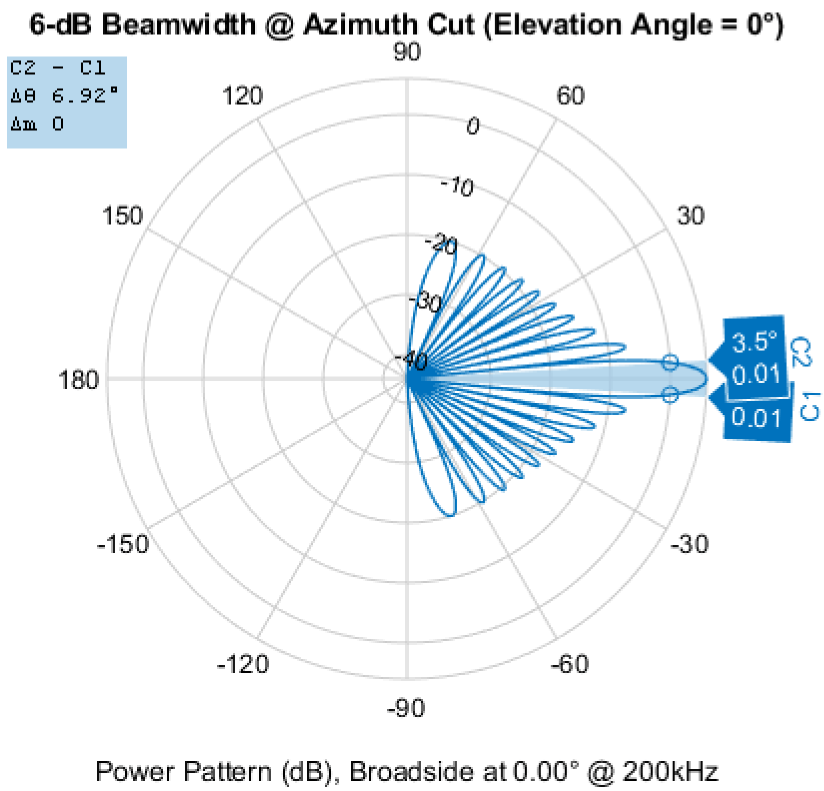
```
myArray = phased.ULA('Element',projector,'NumElements',20,...
    'ElementSpacing',1500/200e3/2)
```

```
myArray = 
  phased.ULA with properties:
```

```
          Element: [1x1 phased.IsotropicProjector]
      NumElements: 20
   ElementSpacing: 0.0037
        ArrayAxis: 'y'
            Taper: 1
```

Using the `beamwidth` function, calculate and plot the 6 dB beamwidth of the sonar array.

```
beamwidth(myArray,200e3,'dBDown',6,'PropagationSpeed',1500)
```




```
ans = 6.9200
```

Calculate Beamwidth and Angles of Uniform Linear Array (ULA)

Calculate the half-power beamwidth and angles of a 20-element uniform linear array (ULA) of cosine antenna elements.

Create a `phased.CosineAntennaElement` object with the `'CosinePower'` exponents set to 1.5.

```
myAnt = phased.CosineAntennaElement
```

```
myAnt = 
  phased.CosineAntennaElement with properties:
```

```
    FrequencyRange: [0 1.0000e+20]
       CosinePower: [1.5000 1.5000]
```

Create a `phased.ULA` object to model a 20-element ULA of cosine antenna elements. These elements are spaced at 0.5 meters on the azimuth plane.

```
array = phased.ULA('Element',myAnt,'NumElements',20)
```

```
array = 
  phased.ULA with properties:

           Element: [1x1 phased.CosineAntennaElement]
       NumElements: 20
    ElementSpacing: 0.5000
         ArrayAxis: 'y'
             Taper: 1
```

Compute the beamwidth and angles of the array when it is operating at 3e8 Hz. Specify the beamwidth to be computed along the elevation plane.

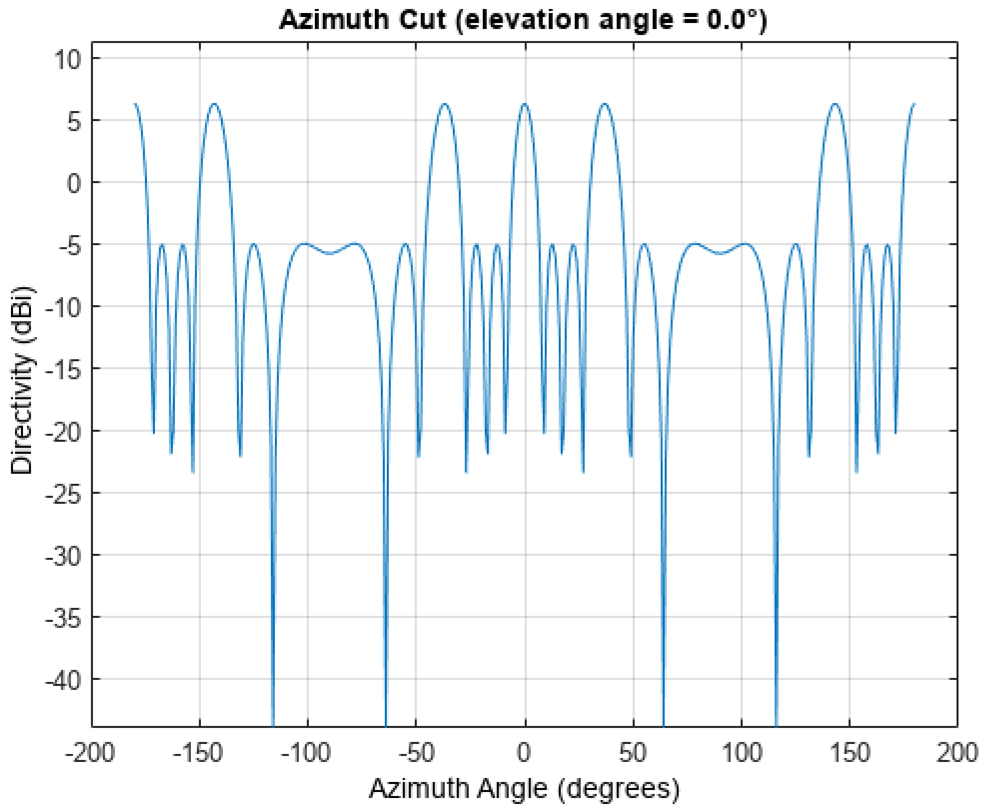
```
[BW,Ang] = beamwidth(array,3e8,'Cut','Elevation')
```

```
BW = 74.8200

Ang = 1×2

  -37.4100   37.4100
```

Response of Antenna ULA

Create a 4-element ULA of isotropic antenna elements and find the response of each element at boresight. Plot the array response at 1 GHz for azimuth angles between -180 and 180 degrees.

```
ha = phased.ULA('NumElements',4);
fc = 1e9;
ang = [0;0];
resp = step(ha,fc,ang);
c = physconst('LightSpeed');
pattern(ha,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','rectangular')
```



Response of Microphone ULA Array

Find the response of a ULA array of 10 omnidirectional microphones spaced 1.5 meters apart. Set the frequency response of the microphone to the range 20 Hz to 20 kHz and choose the signal frequency to be 100 Hz. Using the `step` method, determine the response of each element at boresight: 0 degrees azimuth and 0 degrees elevation.

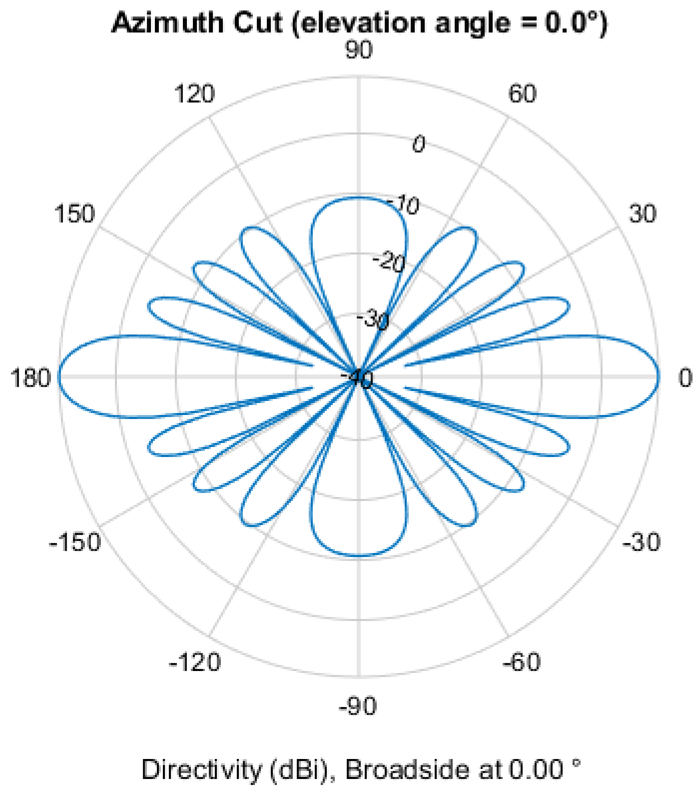
```
mic = phased.OmnidirectionalMicrophoneElement( ...
    'FrequencyRange',[20 20e3]);
Nelem = 10;
array = phased.ULA('NumElements',Nelem, ...
    'ElementSpacing',1.5,'Element',mic);
fc = 100;
ang = [0;0];
resp = array(fc,ang)
```

```
resp = 10×1

     1
     1
     1
     1
     1
     1
     1
```

```
1
1
1
```

Plot the array directivity. Assume the speed of sound in air to be 340 m/s.

```
c = 340;
pattern(array,fc,[-180:180],0,'PropagationSpeed',c,'CoordinateSystem','polar')
```



Simulate Received Signals at ULA

Simulate two received plane-wave random signals at a 4-element ULA. The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
array = phased.ULA(4);
y = collectPlaneWave(array,randn(4,2),[10 30],100e6,physconst('LightSpeed'))
```

```
y = 4×4 complex

   0.7430 - 0.3705i   0.8433 - 0.1314i   0.8433 + 0.1314i   0.7430 + 0.3705i
   0.8418 + 0.4308i   0.5632 + 0.1721i   0.5632 - 0.1721i   0.8418 - 0.4308i
  -2.4817 + 0.9157i  -2.6683 + 0.3175i  -2.6683 - 0.3175i  -2.4817 - 0.9157i
```

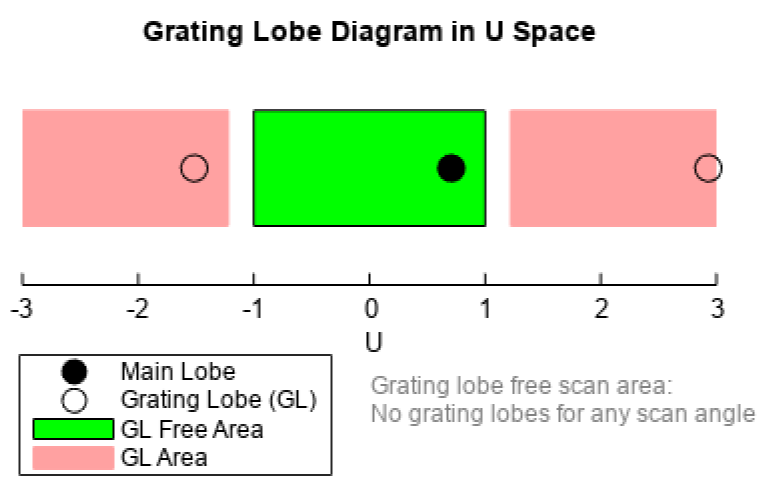
```
1.0724 - 0.4748i   1.1895 - 0.1671i   1.1895 + 0.1671i   1.0724 + 0.4748i
```

Create Grating Lobe Diagram for ULA

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing less than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 3 GHz and the spacing between elements is 0.45 of the wavelength. All elements are isotropic antenna elements. Steer the array in the direction 45 degrees in azimuth and 0 degrees in elevation.

```
c = physconst('LightSpeed');
f = 3e9;
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,...
    'ElementSpacing',0.45*lambda);
plotGratingLobeDiagram(sULA,f,[45;0],c);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region is defined by the direction cosine limits between [-1,1] and is marked by the two vertical black lines. Because the array spacing is less than one-half wavelength, there are no grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it coincides with the visible region.

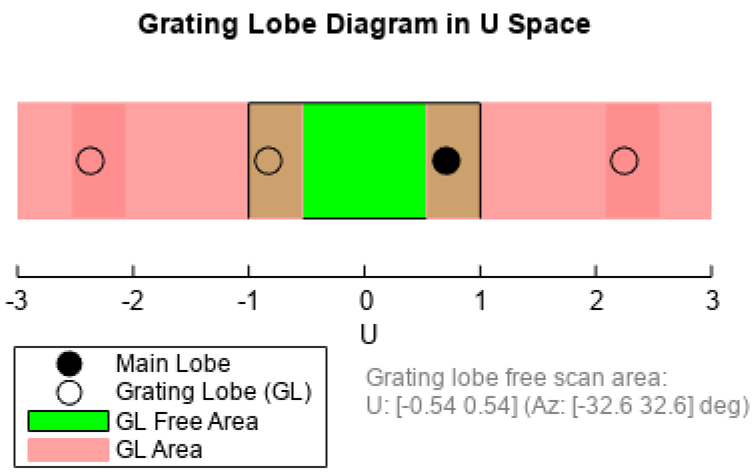
The white area of the diagram indicates a region where no grating lobes are possible.

Create Grating Lobe Diagram for Undersampled ULA

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing greater than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 3 GHz and the spacing between elements is 0.65 of a wavelength. All elements are isotropic antenna elements. Steer the array in the direction 45 degrees in azimuth and 0 degrees in elevation.

```
c = physconst('LightSpeed');
f = 3e9;
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,'ElementSpacing',0.65*lambda);
plotGratingLobeDiagram(sULA,f,[45;0],c);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region, marked by the two black vertical lines, corresponds to arrival angles between -90 and 90 degrees. The visible region is defined by the direction cosine limits $-1 \leq u \leq 1$. Because the array spacing is greater than one-half wavelength, there is now a grating lobe in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those for which $-3 \leq u \leq 3$ are shown.

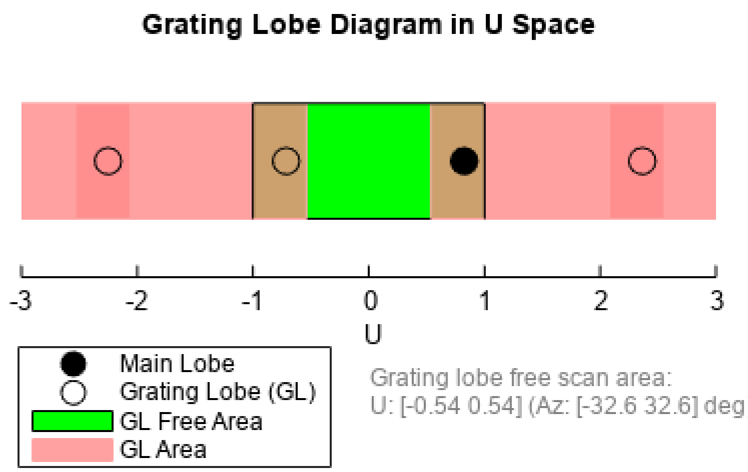
The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region.

Create Grating Lobe Diagram for ULA With Different Phase-Shifter Frequency

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing greater than one-half wavelength. Apply a phase-shifter frequency that differs from the signal frequency. Grating lobes are plotted in u-v coordinates.

Assume the signal frequency is 3 GHz and the spacing between elements is 0.65 λ. All elements are isotropic antenna elements. The phase-shifter frequency is set to 3.5 GHz. Steer the array in the direction 45° azimuth, 0° elevation.

```
c = physconst('LightSpeed');
f = 3e9;
f0 = 3.5e9;
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,...
    'ElementSpacing',0.65*lambda );
plotGratingLobeDiagram(sULA,f,[45;0],c,f0);
```



As a result of adding the shifted frequency, the mainlobe shifts right towards larger *u* values. The beam no longer points toward the actual source arrival angle.

The mainlobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region, marked by the two black vertical lines, corresponds to arrival angles between -90˚ and 90˚. The visible region is defined by the direction cosine limits $-1 \le u \le 1$. Because the array spacing is greater than one-half wavelength, there is now a grating lobe in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those for which $-3 \le u \le 3$ are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region.

Version History

Introduced in R2011a

References

[1] Brookner, E., ed. *Radar Technology*. Lexington, MA: LexBook, 1996.

[2] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, `plotResponse`, and `viewArray` methods are not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also
`phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.CrossedDipoleAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ShortDipoleAntennaElement` | `phased.URA` | `phased.UCA` | `phased.HeterogeneousULA` | `phased.HeterogeneousURA`

Topics
Phased Array Gallery

directivity

System object: phased.ULA
Package: phased

Directivity of uniform linear array

Syntax

```
D = directivity(H,FREQ,ANGLE)
D = directivity(H,FREQ,ANGLE,Name,Value)
```

Description

D = directivity(H,FREQ,ANGLE) computes the "Directivity (dBi)" on page 1-1819 of a uniform linear array (ULA) of antenna or microphone elements, H, at frequencies specified by FREQ and in angles of direction specified by ANGLE.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

D = directivity(H,FREQ,ANGLE,Name,Value) returns the directivity with additional options specified by one or more Name,Value pair arguments.

Input Arguments

H — Uniform linear array

System object

Uniform linear array specified as a phased.ULA System object.

Example: H = phased.ULA;

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.
- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: [1e8 2e6]

Data Types: double

`ANGLE` — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Uniform Linear Array

Compute the directivities of two different uniform linear arrays (ULA). One array consists of isotropic antenna elements and the second array consists of cosine antenna elements. In addition, compute the directivity when the first array is steered in a specified direction. For each case, calculate the directivities for a set of seven different azimuth directions all at zero degrees elevation. Set the frequency to 800 MHz.

Array of isotropic antenna elements

First, create a 10-element ULA of isotropic antenna elements spaced 1/2-wavelength apart.

```
c = physconst('LightSpeed');
fc = 3e8;
lambda = c/fc;
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
myAnt1 = phased.IsotropicAntennaElement;
myArray1 = phased.ULA(10,lambda/2,'Element',myAnt1);
```

Compute the directivity.

```
d = directivity(myArray1,fc,ang,'PropagationSpeed',c)
```

```
d = 7×1

   -6.9886
   -6.2283
   -6.5176
   10.0011
   -6.5176
```

```
   -6.2283
   -6.9886
```

Array of cosine antenna elements

Next, create a 10-element ULA of cosine antenna elements spaced 1/2-wavelength apart.

```
myAnt2 = phased.CosineAntennaElement('CosinePower',[1.8,1.8]);
myArray2 = phased.ULA(10,lambda/2,'Element',myAnt2);
```

Compute the directivity.

```
d = directivity(myArray2,fc,ang,'PropagationSpeed',c)
```

```
d = 7×1

   -1.9838
    0.0529
    0.4968
   17.2548
    0.4968
    0.0529
   -1.9838
```

The directivity of the cosine ULA is greater than the directivity of the isotropic ULA because of the larger directivity of the cosine antenna element.

Steered array of isotropic antenna elements

Finally, steer the isotropic antenna array to 30 degrees in azimuth and compute the directivity.

```
w = steervec(getElementPosition(myArray1)/lambda,[30;0]);
d = directivity(myArray1,fc,ang,'PropagationSpeed',c,...
    'Weights',w)
```

```
d = 7×1

 -297.2705
  -13.9783
   -9.5713
   -6.9897
   -4.5787
   -2.0536
   10.0000
```

The directivity is greatest in the steered direction.

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternAzimuth` | `patternElevation`

collectPlaneWave

System object: phased.ULA
Package: phased

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

`Y = collectPlaneWave(H,X,ANG)` returns the received signals at the sensor array, `H`, when the input signals indicated by `X` arrive at the array from the directions specified in `ANG`.

`Y = collectPlaneWave(H,X,ANG,FREQ)`, in addition, specifies the incoming signal carrier frequency in `FREQ`.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)`, in addition, specifies the signal propagation speed in `C`.

Input Arguments

H

Array object.

X

Incoming signals, specified as an M-column matrix. Each column of `X` represents an individual incoming signal.

ANG

Directions from which incoming signals arrive, in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column specifies the direction of arrival of the corresponding signal in `X`. Each column of `ANG` is in the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `ANG` is a row vector of length M, each entry in `ANG` specifies the azimuth angle. In this case, the corresponding elevation angle is assumed to be 0°.

FREQ

Carrier frequency of signal in hertz. `FREQ` must be a scalar.

Default: `3e8`

c

Propagation speed of signal in meters per second.

Default: Speed of light

Output Arguments

Y

Received signals. Y is an N-column matrix, where N is the number of elements in the array H. Each column of Y is the received signal at the corresponding array element, with all incoming signals combined.

Examples

Simulate Received Signals at ULA

Simulate two received plane-wave random signals at a 4-element ULA. The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
array = phased.ULA(4);
y = collectPlaneWave(array,randn(4,2),[10 30],100e6,physconst('LightSpeed'))
```

```
y = 4×4 complex

   0.7430 - 0.3705i   0.8433 - 0.1314i   0.8433 + 0.1314i   0.7430 + 0.3705i
   0.8418 + 0.4308i   0.5632 + 0.1721i   0.5632 - 0.1721i   0.8418 - 0.4308i
  -2.4817 + 0.9157i  -2.6683 + 0.3175i  -2.6683 - 0.3175i  -2.4817 - 0.9157i
   1.0724 - 0.4748i   1.1895 - 0.1671i   1.1895 + 0.1671i   1.0724 + 0.4748i
```

Algorithms

collectPlaneWave modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. The method does not account for the response of individual elements in the array.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

uv2azel | phitheta2azel

getElementPosition

System object: phased.ULA
Package: phased

Positions of array elements

Syntax

```
pos = getElementPosition(sULA)
pos = getElementPosition(sULA,elemidx)
```

Description

`pos = getElementPosition(sULA)` returns the element positions of the `phased.ULA` System object, `sULA`. `pos` is a 3-by-*N* matrix, where *N* is the number of elements in `sULA`. Each column of `pos` defines the position of an element in the local coordinate system taking the form `[x;y;z]`. Units are meters. The origin of the local coordinate system is the phase center of the array.

`pos = getElementPosition(sULA,elemidx)` returns only the positions of the elements that are specified in the element index vector `elemidx`. This syntax can use any of the input arguments in the previous syntax.

Examples

ULA Element Positions

Construct a ULA with 5 elements along the z-axis. Obtain the element positions.

```
sULA = phased.ULA('NumElements',5,'ArrayAxis','z');
pos = getElementPosition(sULA)
```

```
pos = 3×5

         0         0         0         0         0
         0         0         0         0         0
   -1.0000   -0.5000         0    0.5000    1.0000
```

getElementNormal

System object: phased.ULA
Package: phased

Normal vector to array elements

Syntax

```
normvec = getElementNormal(sULA)
normvec = getElementNormal(sULA,elemidx)
```

Description

normvec = getElementNormal(sULA) returns the normal vectors of the array elements of the phased.ULA System object, sULA. The output argument normvec is a 2-by-*N* matrix, where *N* is the number of elements in array, sULA. Each column of normvec defines the normal direction of an element in the local coordinate system in the form[az;el]. Units are degrees. Array elements are located along the axis selected in the ArrayAxis property. Element normal vectors are parallel to the array normal. The normal to a ULA array depends upon the selected ArrayAxis property.

ArrayAxis Property Value	Array Normal Direction
'x'	azimuth = 90°, elevation = 0° (*y*-axis)
'y'	azimuth = 0°, elevation = 0° (*x*-axis)
'z'	azimuth = 0°, elevation = 0° (*x*-axis)

The origin of the local coordinate system is defined by the phase center of the array.

normvec = getElementNormal(sULA,elemidx) returns only the normal vectors of the elements specified in the element index vector, elemidx. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

sULA — Uniform line array
phased.ULA System object

Uniform line array, specified as a phased.ULA System object.

Example: sULA = phased.ULA

elemidx — Element indices
all array elements (default) | integer-valued 1-by-*M* row vector | integer-valued *M*-by-1 column vector

Element indices , specified as a 1-by-*M* or *M*-by-1 vector. Index values lie in the range 1 to *N* where *N* is the number of elements of the array. When elemidx is specified, getElementNormal returns the normal vectors of the elements contained in elemidx.

Example: [1,5,4]

Output Arguments

normvec — Element normal vectors
2-by-*P* real-valued vector

Element normal vectors, specified as a 2-by-*P* real-valued vector. Each column of `normvec` takes the form `[az,el]`. When `elemidx` is not specified, *P* equals the array dimension. When `elemidx` is specified, *P* equals the length of `elemidx`, *M*.

Examples

ULA Element Normals

Construct three ULAs with elements along the *x*-, *y*-, and *z*-axes. Obtain the element normals.

First, choose the array axis along the *x*-axis.

```
sULA1 = phased.ULA('NumElements',5,'ArrayAxis','x');
norm = getElementNormal(sULA1)
```

```
norm = 2×5

    90    90    90    90    90
     0     0     0     0     0
```

The element normal vectors point along the *y*-axis.

Next, choose the array axis along the *y*-axis.

```
sULA2 = phased.ULA('NumElements',5,'ArrayAxis','y');
norm = getElementNormal(sULA2)
```

```
norm = 2×5

     0     0     0     0     0
     0     0     0     0     0
```

The element normal vectors point along the *x*-axis.

Finally, set the array axis along the *z*-axis. Obtain the normal vectors of the odd-numbered elements.

```
sULA3 = phased.ULA('NumElements',5,'ArrayAxis','z');
norm = getElementNormal(sULA3,[1,3,5])
```

```
norm = 2×3

     0     0     0
     0     0     0
```

The element normal vectors also point along the *x*-axis.

Version History

Introduced in R2016a

getNumElements

System object: `phased.ULA`
Package: `phased`

Number of elements in array

Syntax

`N = getNumElements(H)`

Description

`N = getNumElements(H)` returns the number of elements, N, in the ULA object `H`.

Examples

Get Number of ULA Elements

Construct a default ULA and obtain the number of elements in that array.

```
array = phased.ULA;
N = getNumElements(array)
```

```
N = 2
```

getTaper

System object: phased.ULA
Package: phased

Array element tapers

Syntax

```
wts = getTaper(h)
```

Description

`wts = getTaper(h)` returns the tapers, `wts`, applied to each element of the phased uniform line array (ULA), `h`. Tapers are often referred to as weights.

Input Arguments

h — Uniform line array
phased.ULA System object

Uniform line array specified as a `phased.ULA` System object.

Output Arguments

wts — Array element tapers
N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1 complex-valued vector, where *N* is the number of elements in the array.

Examples

Construct ULA with Taylor Window

Construct a 5-element ULA with a Taylor window taper. Then, obtain the element taper values.

```
taper = taylorwin(5)';
array = phased.ULA(5,'Taper',taper);
w = getTaper(array)
```

```
w = 5×1

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

isPolarizationCapable

System object: phased.ULA
Package: phased

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

flag = isPolarizationCapable(h) returns a Boolean value, flag, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h — Uniform line array

Uniform line array specified as a phased.ULA System object.

Output Arguments

flag — Polarization-capability flag

Polarization-capability flag returned as a Boolean value true if the array supports polarization or false if it does not.

Examples

Short-Dipole Antenna ULA Supports Polarization

Show that an array of phased.ShortDipoleAntennaElement antenna elements supports polarization.

```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[1e9 10e9]);
array = phased.ULA('NumElements',3,'Element',antenna);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

The returned value of 1 shows that this array supports polarization.

plotResponse

System object: `phased.ULA`
Package: `phased`

Plot response pattern of array

Syntax

```
plotResponse(H,FREQ,V)
plotResponse(H,FREQ,V,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ,V)` plots the array response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`. The propagation speed is specified in `V`.

`plotResponse(H,FREQ,V,Name,Value)` plots the array response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

H

Array object

FREQ

Operating frequency in Hertz specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

CutAngle

Cut angle as a scalar. This argument is applicable only when RespCut is 'Az' or 'El'. If RespCut is 'Az', CutAngle must be between –90 and 90. If RespCut is 'El', CutAngle must be between –180 and 180.

Default: 0

Format

Format of the plot, using one of 'Line', 'Polar', or 'UV'. If you set Format to 'UV', FREQ must be a scalar.

Default: 'Line'

NormalizeResponse

Set this value to true to normalize the response pattern. Set this value to false to plot the response pattern without normalizing it. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: true

OverlayFreq

Set this value to true to overlay pattern cuts in a 2-D line plot. Set this value to false to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is false, FREQ must be a vector with at least two entries.

This parameter applies only when Format is not 'Polar' and RespCut is not '3D'.

Default: true

Polarization

Specify the polarization options for plotting the array response pattern. The allowable values are |'None' | 'Combined' | 'H' | 'V' | where

- 'None' specifies plotting a nonpolarized response pattern
- 'Combined' specifies plotting a combined polarization response pattern
- 'H' specifies plotting the horizontal polarization response pattern
- 'V' specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is 'None'. This parameter is not applicable when you set the Unit parameter value to 'dbi'.

Default: 'None'

RespCut

Cut of the response. Valid values depend on Format, as follows:

- If Format is 'Line' or 'Polar', the valid values of RespCut are 'Az', 'El', and '3D'. The default is 'Az'.

- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

`Unit`

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

`Weights`

Weight values applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. The dimension *N* is the number of elements in the array. The interpretation of *M* depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

`AzimuthAngles`

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

`ElevationAngles`

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the

Format parameter is set to 'Line' or 'Polar'. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the RespCut parameter to '3D', you can set the ElevationAngles and AzimuthAngles parameters simultaneously.

Default: [-90:90]

UGrid

U coordinate values for plotting array response, specified as a row vector. The UGrid parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to 'U' or '3D'. The values of UGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set the UGrid and VGrid parameters simultaneously.

Default: [-1:0.01:1]

VGrid

V coordinate values for plotting array response, specified as a row vector. The VGrid parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the Format parameter is set to 'UV' and the RespCut parameter is set to '3D'. The values of VGrid should be between –1 and 1 and should be specified in nondecreasing order. You can set VGrid and UGrid parameters simultaneously.

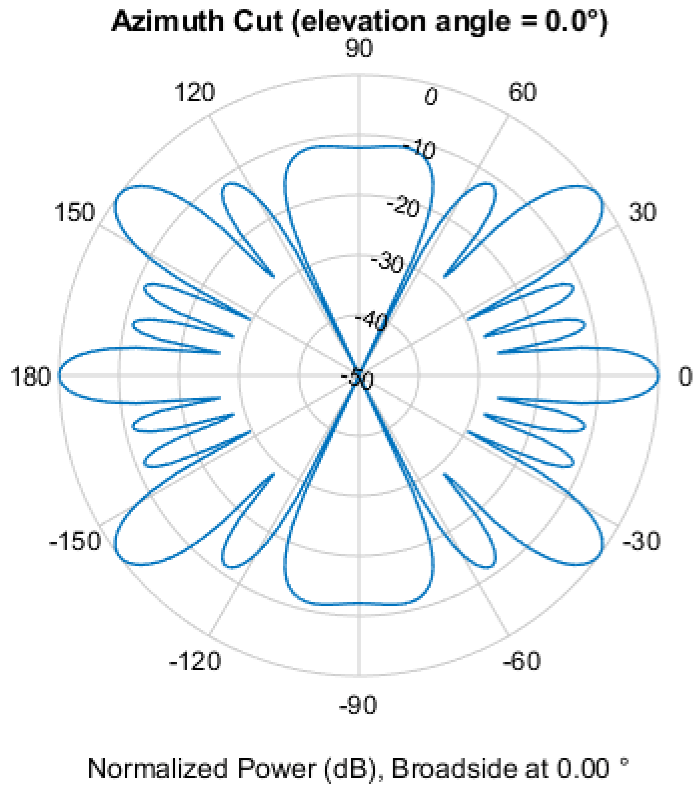
Default: [-1:0.01:1]

Examples

Plot Azimuth Response of 4-Element ULA

Construct a 4-element ULA of isotropic elements (the default) and plot its azimuth response in polar form. By default, the azimuth cut is at 0 degrees elevation. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light. The nominal element spacing is 1/2 meter which means that the array is undersampled at this frequency.

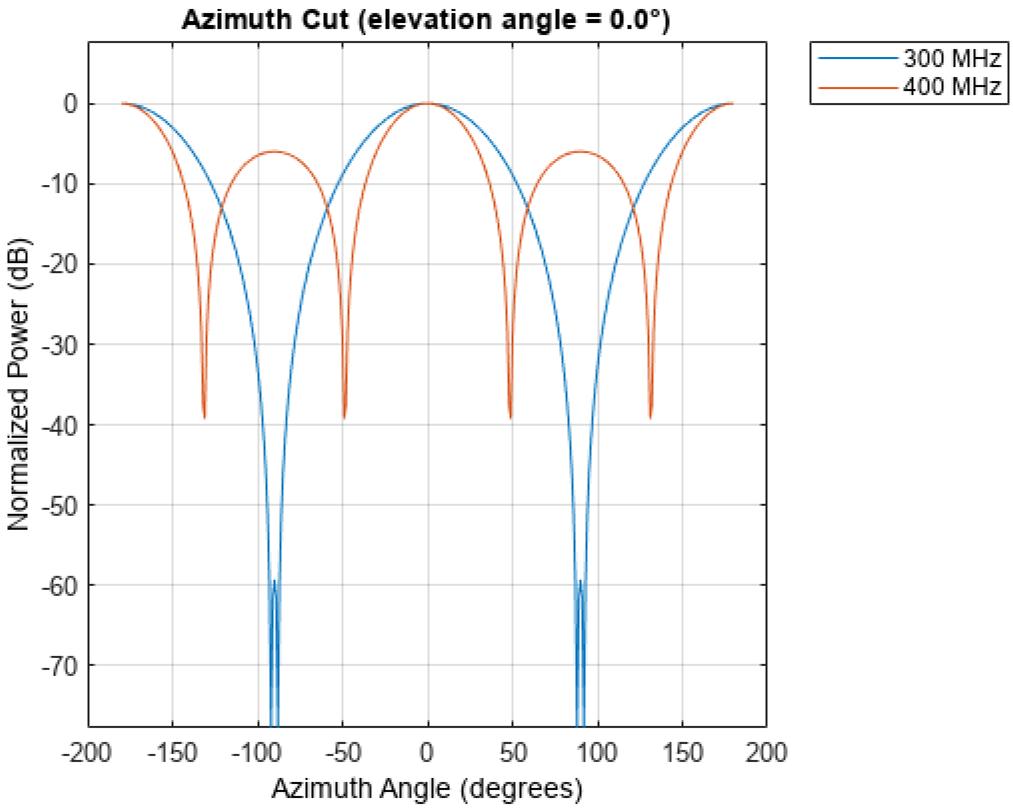
```
ha = phased.ULA(4);
fc = 1e9;
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



Plot Response of ULA at Two Frequencies

This example shows how to plot an azimuth cut of the response of a uniform linear array at 0 degrees elevation using a line plot. The plot shows the responses at operating frequencies of 300 MHz and 400 MHz.

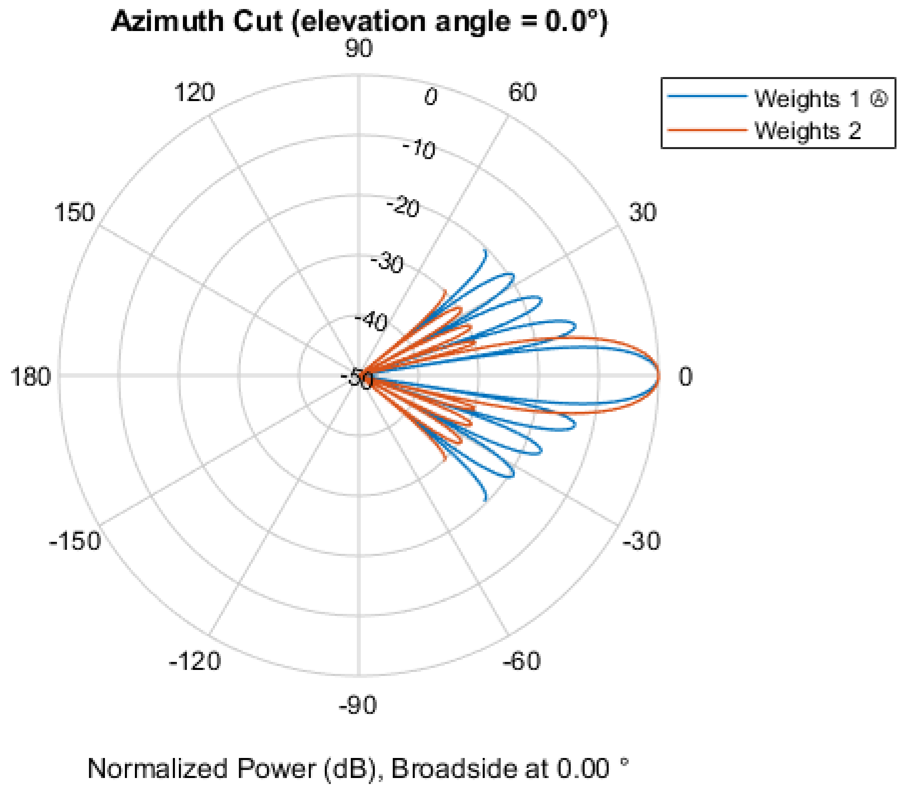
```
h = phased.ULA;
fc = [3e8 4e8];
c = physconst('LightSpeed');
plotResponse(h,fc,c);
```



Plot Azimuth Response of Tapered 11-Element ULA

This example shows how to construct an 11-element ULA array of backbaffled omnidirectional microphones for beamforming the direction of arrival of sound in air. The elements are spaced four centimeters apart and have a frequency response lying in the 2000-8000 Hz frequency range. Use the `plotResponse` method to display an azimuth cut of the array's response at 5000 Hz. Use the `'Weights'` parameter to apply both uniform tapering and Taylor window tapering to the array at the same frequency. Finally, use the `'AzimuthAngles'` parameter to limit the display from -45 to 45 degrees in 0.1 degree increments. A typical value for the speed of sound in air is 343 meters/second.

```
s_omni = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[2000,8000],...
    'BackBaffled',true);
s_ula = phased.ULA(11,'Element',s_omni,...
    'ElementSpacing',0.04);
c = 343.0;
fc = 5000;
wts = taylorwin(11);
plotResponse(s_ula,fc,c,'RespCut','Az',...
    'Format','Polar',...
    'Weights',[ones(11,1),wts],...
    'AzimuthAngles',[-45:.1:45]);
```

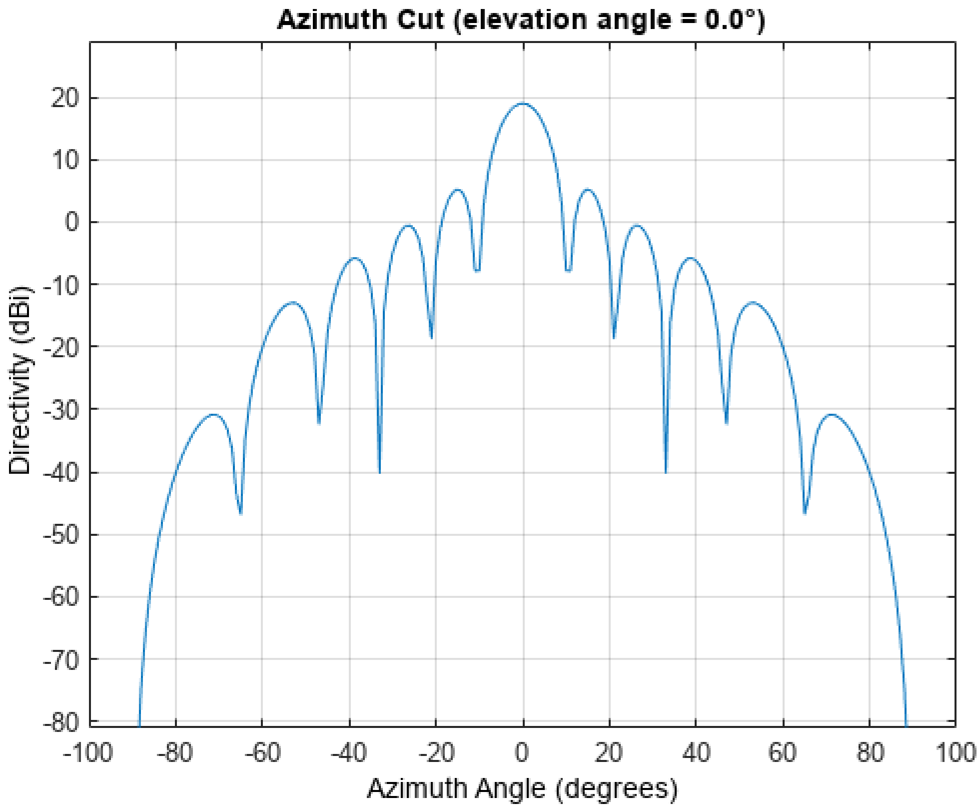


The plot shows that the Taylor tapered set of weights reduces the adjacent sidelobes while broadening the main lobe compared to a uniformly tapered array.

Plot Directivity of 11-Element ULA of Cosine Pattern Antennas

This example shows how to construct an 11-element ULA of cosine antenna elements that are spaced one-half wavelength apart. Then, using the `plotResponse` method, plot an azimuth cut of the array's directivity by setting the `'Unit'` parameter to `'dbi'`. Assume the operating frequency is 1.5 GHz and the wave propagation speed is the speed of light.

```
fc = 1.5e9;
c = physconst('Lightspeed');
lambda = c/fc;
sCos = phased.CosineAntennaElement('FrequencyRange',...
    [1e9 2e9],'CosinePower',[2.5,3.5]);
sULA = phased.ULA(11,0.5*lambda,'Element',sCos);
plotResponse(sULA,fc,c,'RespCut','Az','Unit','dbi');
```



See Also

`uv2azel` | `azel2uv`

pattern

System object: `phased.ULA`
Package: `phased`

Plot array pattern

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

`pattern(sArray,FREQ)` plots the 3-D array directivity pattern (in dBi) for the array specified in `sArray`. The operating frequency is specified in `FREQ`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`pattern(sArray,FREQ,AZ)` plots the array directivity pattern at the specified azimuth angle.

`pattern(sArray,FREQ,AZ,EL)` plots the array directivity pattern at specified azimuth and elevation angles.

`pattern( ___ ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`[PAT,AZ_ANG,EL_ANG] = pattern( ___ )` returns the array pattern in `PAT`. The `AZ_ANG` output contains the coordinate values corresponding to the rows of `PAT`. The `EL_ANG` output contains the coordinate values corresponding to the columns of `PAT`. If the `'CoordinateSystem'` parameter is set to `'uv'`, then `AZ_ANG` contains the *U* coordinates of the pattern and `EL_ANG` contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the `plotResponse` method. See "Convert plotResponse to pattern" on page 1-1847 for guidelines on how to use `pattern` in place of `plotResponse`.

Input Arguments

`sArray` — Uniform linear array
System object

Uniform linear array, specified as a `phased.ULA` System object.

Example: `sArray= phased.ULA;`

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Orientation` — Array orientation
`[0;0;0]`. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

`Normalize` — Display normalize pattern
`true` (default) | `false`

Display normalized pattern, specified as the comma-separated pair consisting of `'Normalize`' and a Boolean. Set this parameter to `true` to display a normalized pattern. This parameter does not apply when you set `'Type'` to `'directivity'`. Directivity patterns are already normalized.

Data Types: `logical`

`ShowArray` — View array geometry
`false` (default) | `true`

View the array geometry along with the 3D radiation pattern, specified as `false` or `true`.

Data Types: `logical`

`ShowLocalCoordinates` — Show local coordinate axes
`true` (default) | `false`

Show the local coordinate axes, specified as `true` or `false`.

Data Types: `logical`

`ShowColorbar` — Show colorbar
`true` (default) | `false`

Show the colorbar, specified as `true` or `false`.

Data Types: `logical`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

`PlotStyle` — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

`Polarization` — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

`'Polarization'`	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights`' and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

PAT — Array pattern
M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of `PAT` correspond to the dimensions of the output arguments `AZ_ANG` and `EL_ANG`.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in `AZ`. The columns of `PAT` correspond to the values in `AZ_ANG`. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

Examples

Plot Pattern of 9-Element ULA Antenna Array of Short Dipoles

Create an 9-element ULA of short dipole antenna elements spaced 0.2 meters apart. Display the azimuth and elevation directivities. The operating frequency is 500 MHz. Plot the directivities in polar coordinates.

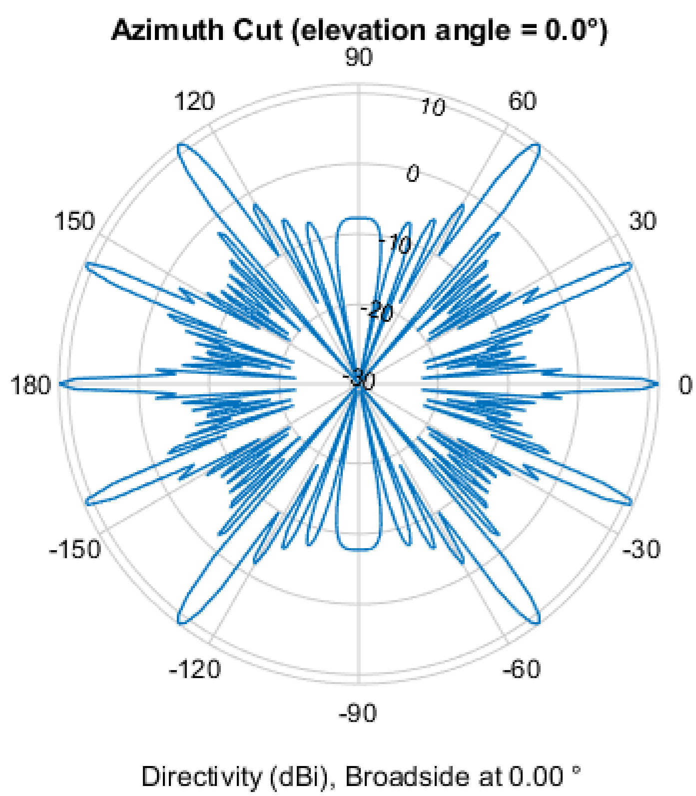
Evaluate the fields at 45 degrees azimuth and 0 degrees elevation.

```
element = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[50e6,1000e6],...
    'AxisDirection','Z');
array = phased.ULA('NumElements',9,'ElementSpacing',1.5,'Element',element);
fc = 500e6;
ang = [45;0];
resp = array(fc,ang);
disp(resp.V)
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

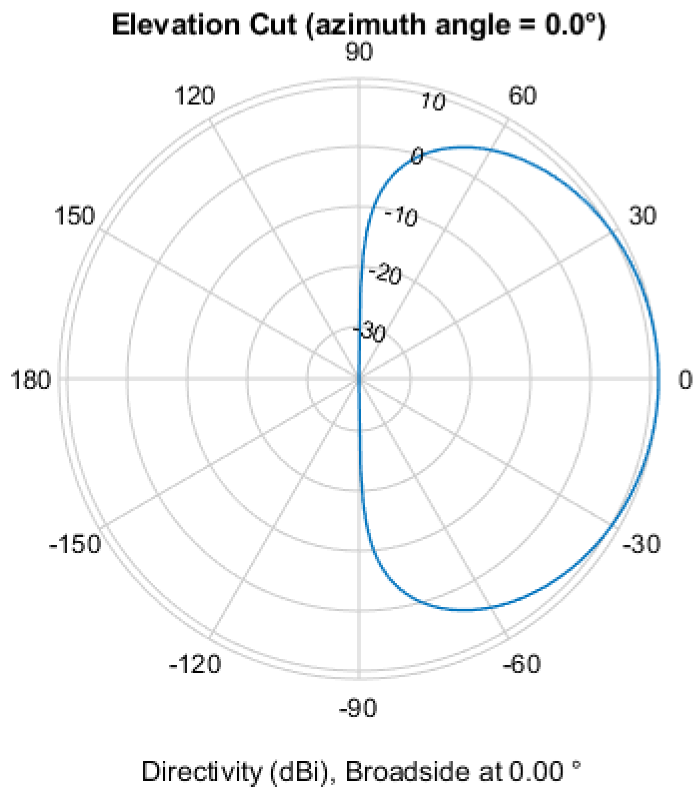
Display the azimuth directivity pattern at 500 MHz for azimuth angles between -180 and 180 degrees.

```
c = physconst('LightSpeed');
pattern(array,fc,[-180:180],0,...
    'Type','directivity',...
    'PropagationSpeed',c)
```



Display the elevation directivity pattern at 500 MHz for elevation angles between -90 and 90 degrees.

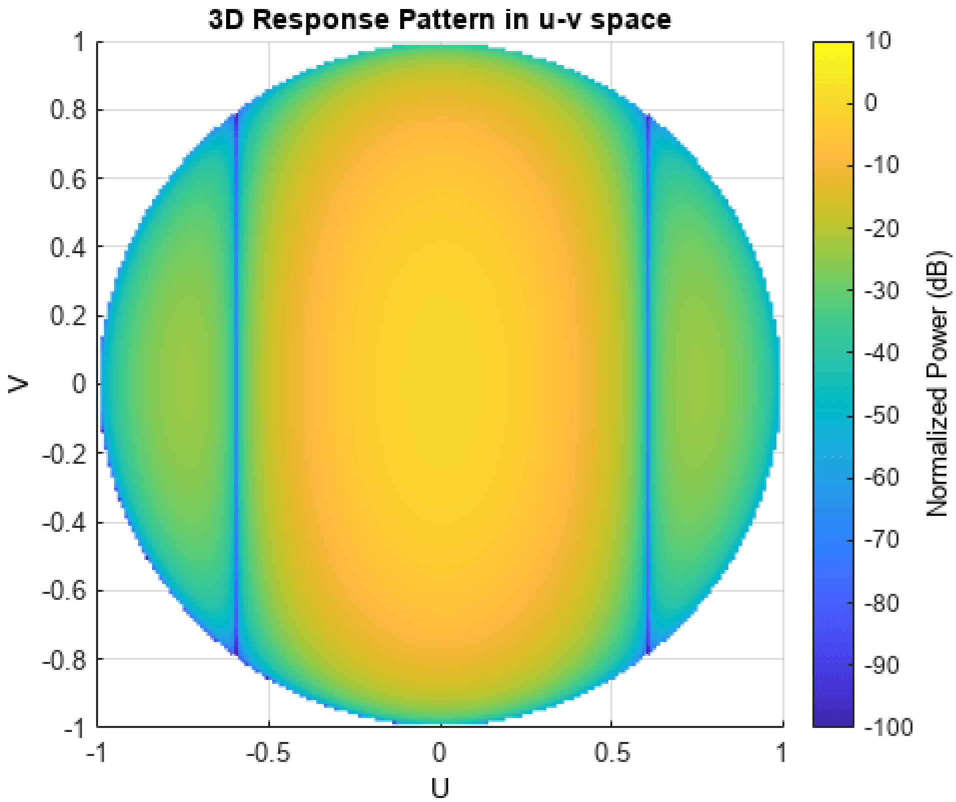
```
pattern(array,fc,[0],[-90:90],...
    'Type','directivity',...
    'PropagationSpeed',c)
```



Plot Pattern of 10-Element ULA Antenna Array in UV Space

Create a 10-element ULA antenna array consisting of cosine antenna elements spaced 10 cm apart. Display the 3-D power pattern in UV space. The operating frequency is 500 MHz.

```
sCos = phased.CosineAntennaElement('FrequencyRange',[100e6 1e9],...
    'CosinePower',[2.5,2.5]);
sULA = phased.ULA('NumElements',10,...
    'ElementSpacing',.1,...
    'Element',sCos);
c = physconst('LightSpeed');
fc = 500e6;
pattern(sULA,fc,[-1:.01:1],[-1:.01:1],...
    'CoordinateSystem','uv',...
    'Type','powerdb',...
    'PropagationSpeed',c)
```



More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These are the `azimuthPattern` and `elevationPattern` methods.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. You should notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and vice versa. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td><code>'Format'</code> and <code>'RespCut'</code> name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using <code>plotResponse</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'RespCut'</code> to <code>'Az'</code> or <code>'El'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using either the <code>'AzimuthAngles'</code> or <code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'RespCut'</code> to <code>'3D'</code>. Set <code>'Format'</code> to <code>'line'</code> or <code>'polar'</code>.

Set the display axis using both the <code>'AzimuthAngles'</code> and<code>'ElevationAngles'</code> name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'RespCut'</code> to<code>'U'</code>. Set <code>'Format'</code> to <code>'UV'</code>. Set the display range using the <code>'UGrid'</code> name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'RespCut'</code> to<code>'3D'</code>. Set <code>'Format'</code> to</td></tr>
</table>
</td>
<td><code>'CoordinateSystem'</code> name-value pair used together with the <code>AZ</code> and <code>EL</code> input arguments.

<code>'CoordinateSystem'</code> has the same options as the <code>plotResponse</code> method <code>'Format'</code>name-value pair, except that <code>'line'</code> is now named <code>'rectangular'</code>. The table shows how to create different types of plots using <code>pattern</code>.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify either <code>AZ</code> or <code>EL</code> as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'rectangular'</code> or <code>'polar'</code>. Specify both <code>AZ</code> and <code>EL</code> as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set <code>'CoordinateSystem'</code> to <code>'uv'</code>. Use <code>AZ</code> to specify a U-space vector. Use <code>EL</code> to specify a V-space vector.</td></tr>
</table>
If you set <code>CoordinateSystem</code> to <code>'uv'</code>, enter the UV grid values using <code>AZ</code> and <code>EL</code>.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
'ElevationAngles' name-value pair	Elevation angles used to display the antenna or array response.	EL argument
'UGrid' name-value pair	Contains *U* coordinates in *UV*-space.	AZ argument when 'CoordinateSystem' name-value pair is set to 'uv'
'VGrid' name-value pair	Contains *V*-coordinates in *UV*-space.	EL argument when 'CoordinateSystem' name-value pair is set to 'uv'

Version History

Introduced in R2015a

See Also

patternAzimuth | patternElevation

patternAzimuth

System object: `phased.ULA`
Package: `phased`

Plot ULA array directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sArray,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(sArray,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sArray,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sArray` — Uniform linear array

System object

Uniform linear array, specified as a `phased.ULA` System object.

Example: `sArray= phased.ULA;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of 'Weights' and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the phased.SteeringVector System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as phased.Radiator or phased.Collector. However, for the directivity, pattern, patternAzimuth, and patternElevation methods of any array System object use the steering vector without conjugation.

Example: 'Weights',ones(10,1)

Data Types: double
Complex Number Support: Yes

Azimuth — Azimuth angles
[-180:180] (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of 'Azimuth' and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: 'Azimuth',[-90:2:90]

Data Types: double

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the 'Azimuth' name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the EL input argument.

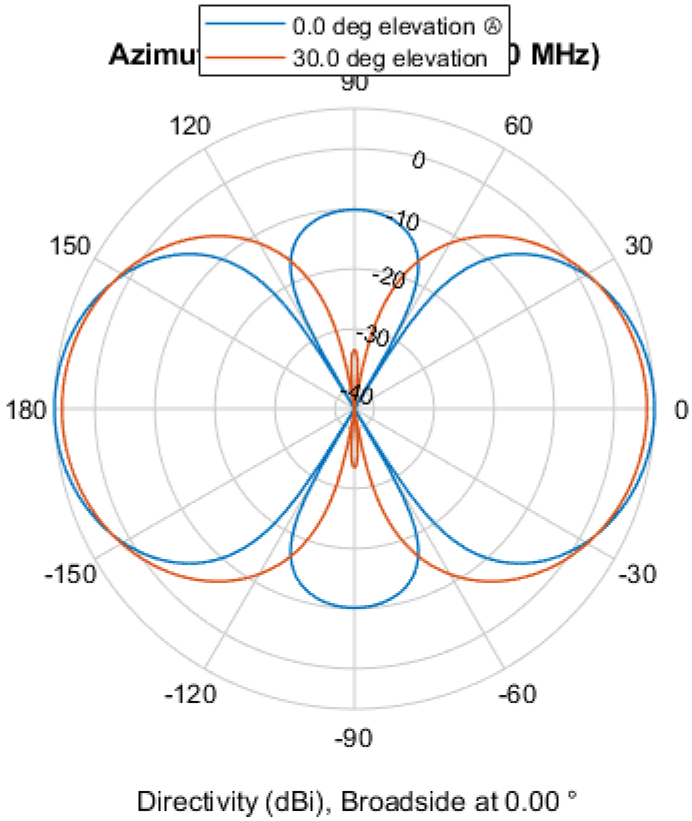
Examples

Plot Azimuth Pattern of ULA

Create a 7-element ULA of short-dipole antenna elements spaced 10 cm apart. Plot an azimuth cut of directivity at 0 and 10 degrees elevation. Assume the operating frequency is 500 MHz.

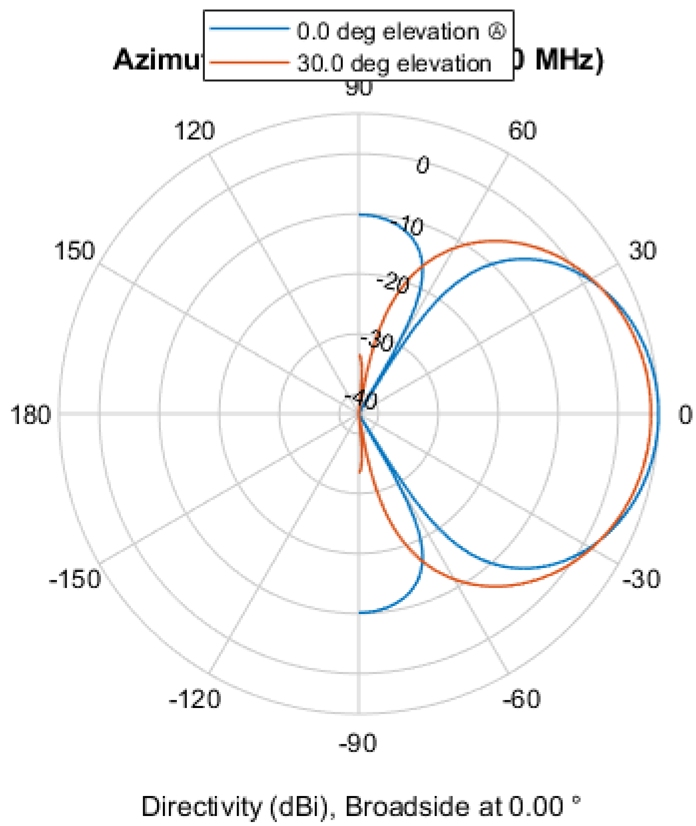
```
fc = 500e6;
sCDant = phased.ShortDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
```

```
sULA = phased.ULA('NumElements',7,'ElementSpacing',0.1,'Element',sCDant);
patternAzimuth(sULA,fc,[0 30])
```



You can plot a smaller range of azimuth angles by setting the `Azimuth` property.

```
patternAzimuth(sULA,fc,[0 30],'Azimuth',[-90:90])
```



More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternElevation`

patternElevation

System object: phased.ULA
Package: phased

Plot ULA array directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

patternElevation(sArray,FREQ) plots the 2-D array directivity pattern versus elevation (in dBi) for the array sArray at zero degrees azimuth angle. When AZ is a vector, multiple overlaid plots are created. The argument FREQ specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

patternElevation(sArray,FREQ,AZ), in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by AZ. When AZ is a vector, multiple overlaid plots are created.

patternElevation(sArray,FREQ,AZ,Name,Value) plots the array pattern with additional options specified by one or more Name,Value pair arguments.

PAT = patternElevation(___) returns the array pattern. PAT is a matrix whose entries represent the pattern at corresponding sampling points specified by the 'Elevation' parameter and the AZ input argument.

Input Arguments

sArray — Uniform linear array
System object

Uniform linear array, specified as a phased.ULA System object.

Example: sArray= phased.ULA;

FREQ — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

`AZ` — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

`PAT` — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

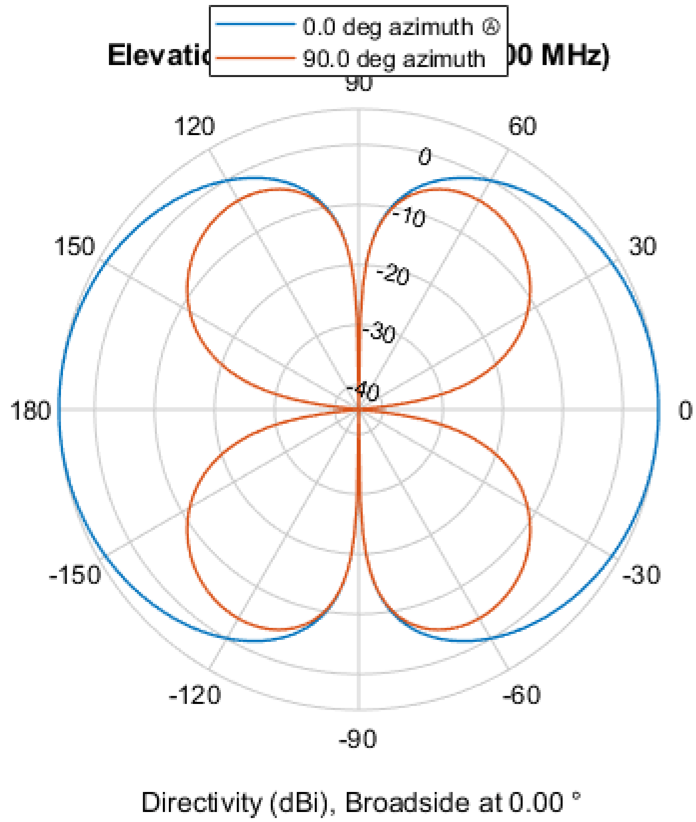
Examples

Plot Elevation Pattern of ULA

Create a 6-element ULA of short-dipole antenna elements with element spacing of 10 cm. Plot an elevation cut of directivity at 0 and 90 degrees azimuth. Assume the operating frequency is 500 MHz.

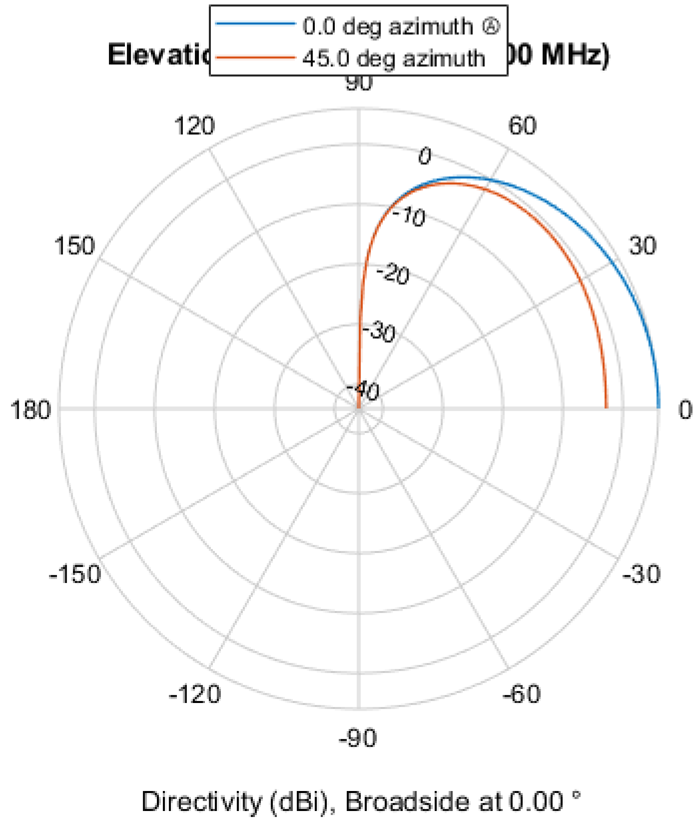
```
fc = 500e6;
c = physconst('LightSpeed');
```

```
sSD = phased.ShortDipoleAntennaElement('FrequencyRange',[100,900]*1e6);
sULA = phased.ULA('NumElements',6,'ElementSpacing',0.1,'Element',sSD);
patternElevation(sULA,fc,[0 90],'PropagationSpeed',c)
```



You can plot a smaller range of elevation angles by setting the `Elevation` property.

```
patternElevation(sULA,fc,[0 45],'Elevation',[0:90],'PropagationSpeed',c)
```



More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also

`pattern` | `patternAzimuth`

plotGratingLobeDiagram

System object: `phased.ULA`
Package: `phased`

Plot grating lobe diagram of array

Syntax

```
plotGratingLobeDiagram(H,FREQ)
plotGratingLobeDiagram(H,FREQ,ANGLE)
plotGratingLobeDiagram(H,FREQ,ANGLE,C)
plotGratingLobeDiagram(H,FREQ,ANGLE,C,F0)
hPlot = plotGratingLobeDiagram( ___ )
```

Description

`plotGratingLobeDiagram(H,FREQ)` plots the grating lobe diagram of an array in the *u-v* coordinate system. The System object `H` specifies the array. The argument `FREQ` specifies the signal frequency and phase-shifter frequency. The array, by default, is steered to 0° azimuth and 0° elevation.

A grating lobe diagram displays the positions of the peaks of the narrowband array pattern. The array pattern depends only upon the geometry of the array and not upon the types of elements which make up the array. Visible and nonvisible grating lobes are displayed as open circles. Only grating lobe peaks near the location of the mainlobe are shown. The mainlobe itself is displayed as a filled circle.

`plotGratingLobeDiagram(H,FREQ,ANGLE)`, in addition, specifies the array steering angle, `ANGLE`.

`plotGratingLobeDiagram(H,FREQ,ANGLE,C)`, in addition, specifies the propagation speed by `C`.

`plotGratingLobeDiagram(H,FREQ,ANGLE,C,F0)`, in addition, specifies an array phase-shifter frequency, `F0`, that differs from the signal frequency, `FREQ`. This argument is useful when the signal no longer satisfies the narrowband assumption and, allows you to estimate the size of beam squint.

`hPlot = plotGratingLobeDiagram( ___ )` returns the handle to the plot for any of the input syntax forms.

Input Arguments

`H`

Antenna or microphone array, specified as a System object.

`FREQ`

Signal frequency, specified as a scalar. Frequency units are hertz. Values must lie within a range specified by the frequency property of the array elements contained in `H.Element`. The frequency property is named `FrequencyRange` or `FrequencyVector`, depending on the element type.

ANGLE

Array steering angle, specified as either a 2-by-1 vector or a scalar. If `ANGLE` is a vector, it takes the form `[azimuth;elevation]`. The azimuth angle must lie in the range `[-180°,180°]`. The elevation angle must lie in the range `[-90°,90°]`. All angle values are specified in degrees. If the argument `ANGLE` is a scalar, it specifies only the azimuth angle where the corresponding elevation angle is 0°.

Default: `[0;0]`

C

Signal propagation speed, specified as a scalar. Units are meters per second.

Default: Speed of light in vacuum

F0

Phase-shifter frequency of the array, specified as a scalar. Frequency units are hertz When this argument is omitted, the phase-shifter frequency is assumed to be the signal frequency, `FREQ`.

Default: `FREQ`

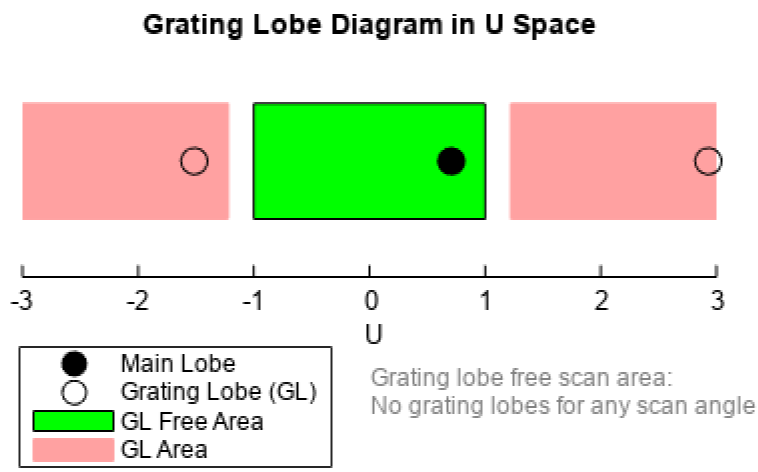
Examples

Create Grating Lobe Diagram for ULA

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing less than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 3 GHz and the spacing between elements is 0.45 of the wavelength. All elements are isotropic antenna elements. Steer the array in the direction 45 degrees in azimuth and 0 degrees in elevation.

```
c = physconst('LightSpeed');
f = 3e9;
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,...
    'ElementSpacing',0.45*lambda);
plotGratingLobeDiagram(sULA,f,[45;0],c);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region is defined by the direction cosine limits between [-1,1] and is marked by the two vertical black lines. Because the array spacing is less than one-half wavelength, there are no grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it coincides with the visible region.

The white area of the diagram indicates a region where no grating lobes are possible.

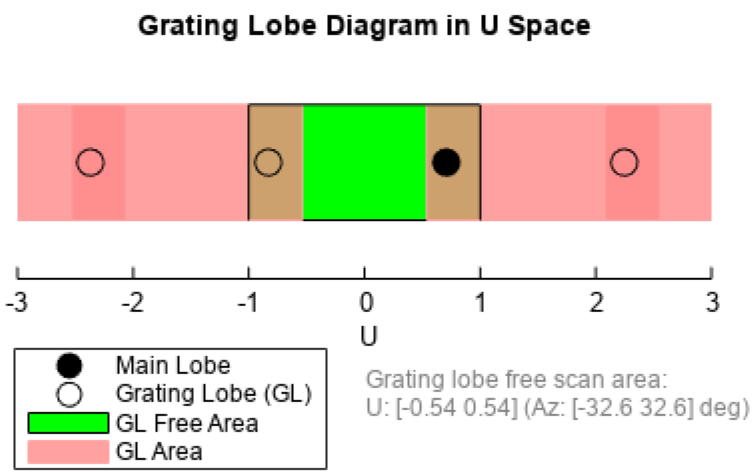
Create Grating Lobe Diagram for Undersampled ULA

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing greater than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 3 GHz and the spacing between elements is 0.65 of a wavelength. All elements are isotropic antenna elements. Steer the array in the direction 45 degrees in azimuth and 0 degrees in elevation.

```
c = physconst('LightSpeed');
f = 3e9;
lambda = c/f;
```

```
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,'ElementSpacing',0.65*lambda);
plotGratingLobeDiagram(sULA,f,[45;0],c);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region, marked by the two black vertical lines, corresponds to arrival angles between -90 and 90 degrees. The visible region is defined by the direction cosine limits $-1 \leq u \leq 1$. Because the array spacing is greater than one-half wavelength, there is now a grating lobe in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those for which $-3 \leq u \leq 3$ are shown.

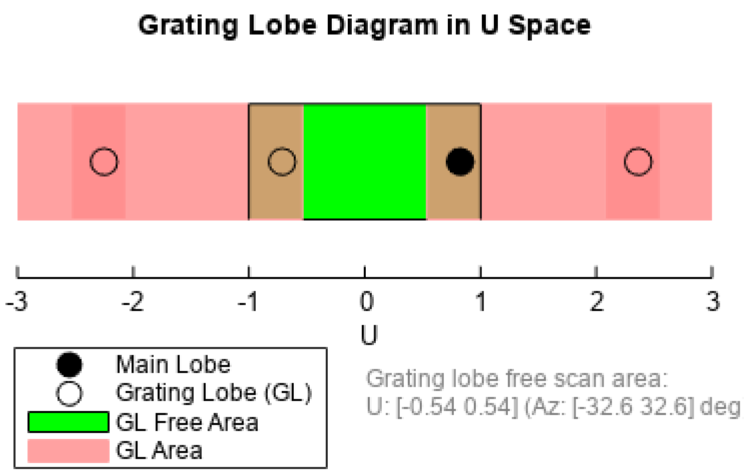
The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region.

Create Grating Lobe Diagram for ULA With Different Phase-Shifter Frequency

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing greater than one-half wavelength. Apply a phase-shifter frequency that differs from the signal frequency. Grating lobes are plotted in u-v coordinates.

Assume the signal frequency is 3 GHz and the spacing between elements is 0.65 λ. All elements are isotropic antenna elements. The phase-shifter frequency is set to 3.5 GHz. Steer the array in the direction 45° azimuth, 0° elevation.

```
c = physconst('LightSpeed');
f = 3e9;
f0 = 3.5e9;
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,...
    'ElementSpacing',0.65*lambda );
plotGratingLobeDiagram(sULA,f,[45;0],c,f0);
```



As a result of adding the shifted frequency, the mainlobe shifts right towards larger *u* values. The beam no longer points toward the actual source arrival angle.

The mainlobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region, marked by the two black vertical lines, corresponds to arrival angles between -90° and 90°. The visible region is defined by the direction cosine limits $-1 \leq u \leq 1$. Because the array spacing is greater than one-half wavelength, there is now a grating lobe in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those for which $-3 \leq u \leq 3$ are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region.

Concepts

Grating Lobes

Spatial undersampling of a wavefield by an array gives rise to visible grating lobes. If you think of the wavenumber, k, as analogous to angular frequency, then you must sample the signal at spatial intervals smaller than π/k_{max} (or $\lambda_{min}/2$) in order to remove aliasing. The appearance of visible grating lobes is also known as spatial aliasing. The variable k_{max} is the largest wavenumber value present in the signal.

The directions of maximum spatial response of a ULA are determined by the peaks of the array's array pattern (alternatively called the beam pattern or array factor). Peaks other than the mainlobe peak are called grating lobes. For a ULA, the array pattern depends only on the wavenumber component of the wavefield along the array axis (the y-direction for the `phased.ULA` System object). The wavenumber component is related to the look-direction of an arriving wavefield by $k_y = -2\pi \sin \varphi/\lambda$. The angle φ is the broadside angle—the angle that the look-direction makes with a plane perpendicular to the array. The look-direction points away from the array to the wavefield source.

The array pattern possesses an infinite number of periodically-spaced peaks that are equal in strength to the mainlobe peak. If you steer the array to the φ_0 direction, the array pattern for a ULA has its mainlobe peak at the wavenumber value of $k_{y0} = -2\pi \sin \varphi_0/\lambda$. The array pattern has strong grating lobe peaks at $k_{ym} = k_{y0} + 2\pi m/d$, for any integer value m. Expressed in terms of direction cosines, the grating lobes occur at $u_m = u_0 + m\lambda/d$, where $u_0 = \sin \varphi_0$. The direction cosine, u_0, is the cosine of the angle that the look-direction makes with the y-axis and is equal to $\sin \varphi_0$ when expressed in terms of the look-direction.

In order to correspond to a physical look-direction, u_m must satisfy, $-1 \le u_m \le 1$. You can compute a physical look-direction angle φ_m from $\sin \varphi_m = u_m$ as long as $-1 \le u_m \le 1$. The spacing of grating lobes depends upon λ/d. When λ/d is small enough, multiple grating lobe peaks can correspond to physical look-directions.

The presence or absence of visible grating lobes for the ULA is summarized in this table.

Element Spacing	Grating Lobes
$\lambda/d \ge 2$	No visible grating lobes for any mainlobe direction.
$1 \le \lambda/d < 2$	Visible grating lobes can exist for some range of mainlobe directions.
$\lambda/d < 1$	Visible grating lobes exist for every mainlobe direction.

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `azel2uv`

step

System object: `phased.ULA`
Package: `phased`

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the array element responses, `RESP`, at the operating frequencies specified in `FREQ` and in directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`H`

Array object

`FREQ`

Operating frequencies of array in hertz. `FREQ` is a row vector of length L. Typical values are within the range specified by a property of `H.Element`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has zero response at frequencies outside that range.

`ANG`

Directions in degrees. `ANG` is either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

If ANG is a row vector of length *M*, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be 0°.

Output Arguments

RESP

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

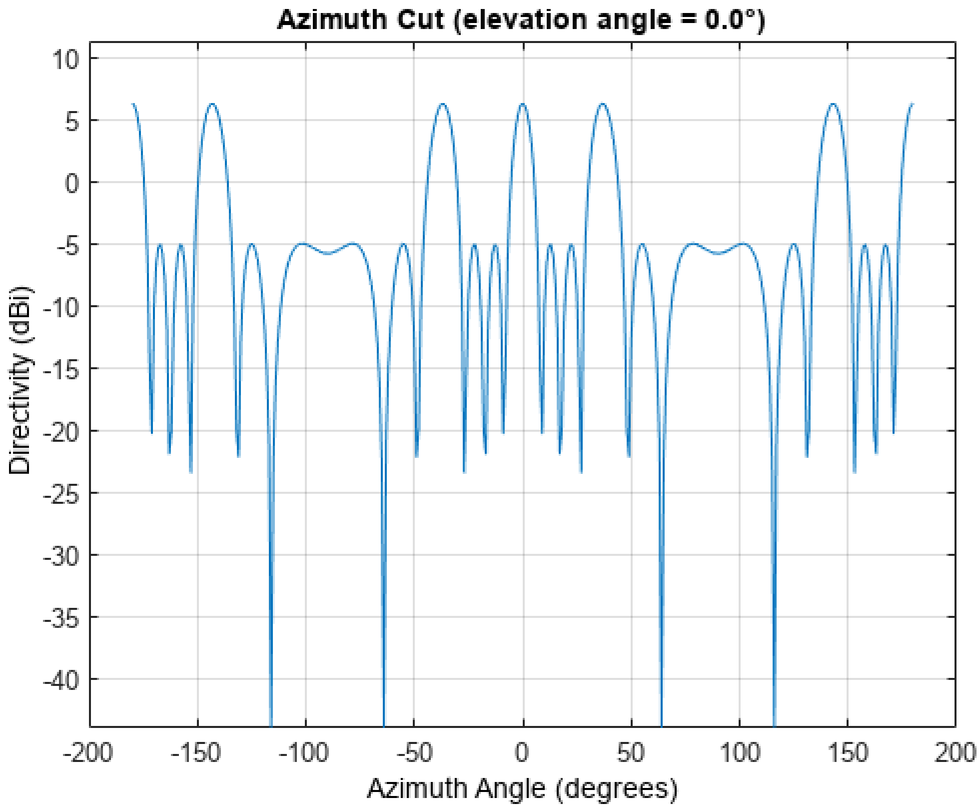
- If the array is not capable of supporting polarization, the voltage response, RESP, has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array. The dimension *M* is the number of angles specified in ANG. *L* is the number of frequencies specified in FREQ. For any element, the columns of RESP contain the responses of the array elements for the corresponding direction specified in ANG. Each of the *L* pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.
- If the array is capable of supporting polarization, the voltage response, RESP, is a MATLAB struct containing two fields, RESP.H and RESP.V. The field, RESP.H, represents the array's horizontal polarization response, while RESP.V represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array, and *M* is the number of angles specified in ANG. *L* is the number of frequencies specified in FREQ. Each column of RESP contains the responses of the array elements for the corresponding direction specified in ANG. Each of the *L* pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.

Examples

Response of Antenna ULA

Create a 4-element ULA of isotropic antenna elements and find the response of each element at boresight. Plot the array response at 1 GHz for azimuth angles between -180 and 180 degrees.

```
ha = phased.ULA('NumElements',4);
fc = 1e9;
ang = [0;0];
resp = step(ha,fc,ang);
c = physconst('LightSpeed');
pattern(ha,fc,[-180:180],0,...
    'PropagationSpeed',c,...
    'CoordinateSystem','rectangular')
```



Response of Microphone ULA Array

Find the response of a ULA array of 10 omnidirectional microphones spaced 1.5 meters apart. Set the frequency response of the microphone to the range 20 Hz to 20 kHz and choose the signal frequency to be 100 Hz. Using the `step` method, determine the response of each element at boresight: 0 degrees azimuth and 0 degrees elevation.

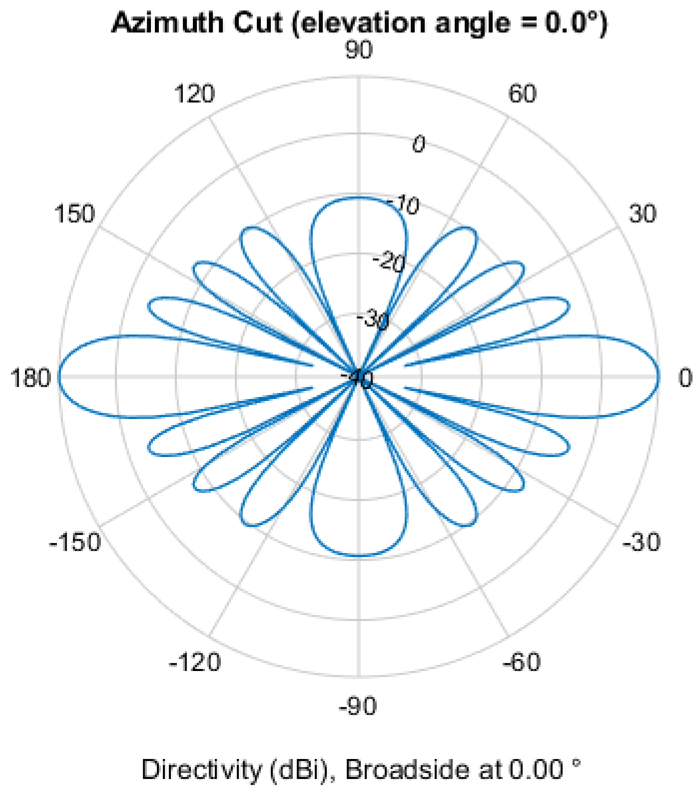
```
mic = phased.OmnidirectionalMicrophoneElement( ...
    'FrequencyRange',[20 20e3]);
Nelem = 10;
array = phased.ULA('NumElements',Nelem, ...
    'ElementSpacing',1.5,'Element',mic);
fc = 100;
ang = [0;0];
resp = array(fc,ang)
```

```
resp = 10×1

     1
     1
     1
     1
     1
     1
     1
```

```
1
1
1
```

Plot the array directivity. Assume the speed of sound in air to be 340 m/s.

```
c = 340;
pattern(array,fc,[-180:180],0,'PropagationSpeed',c,'CoordinateSystem','polar')
```



See Also

uv2azel | phitheta2azel

viewArray

System object: phased.ULA
Package: phased

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

viewArray(H) plots the geometry of the array specified in H.

viewArray(H,Name,Value) plots the geometry of the array, with additional options specified by one or more Name,Value pair arguments.

hPlot = viewArray(___) returns the handle of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

H

Array object.

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Parent

Handle to the axes along which the array geometry is displayed.

ShowNormals

Set this value to true to show the normal directions of all elements of the array. Set this value to false to plot the elements without showing normal directions.

Default: false

ShowLocalCoordinates

Logical flag specifying whether to show the local coordinate axes.

Default: `true`

`ShowAnnotation`

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Default: `true`

`Orientation`

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Default: `[0;0;0]`

`ShowTaper`

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

`ShowIndex`

Vector specifying the element indices to show in the figure. Each number in the vector must be an integer between 1 and the number of elements. You can also specify the value `'All'` to show the indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

`Title`

Character vector specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

`hPlot`

Handle of array elements in figure window.

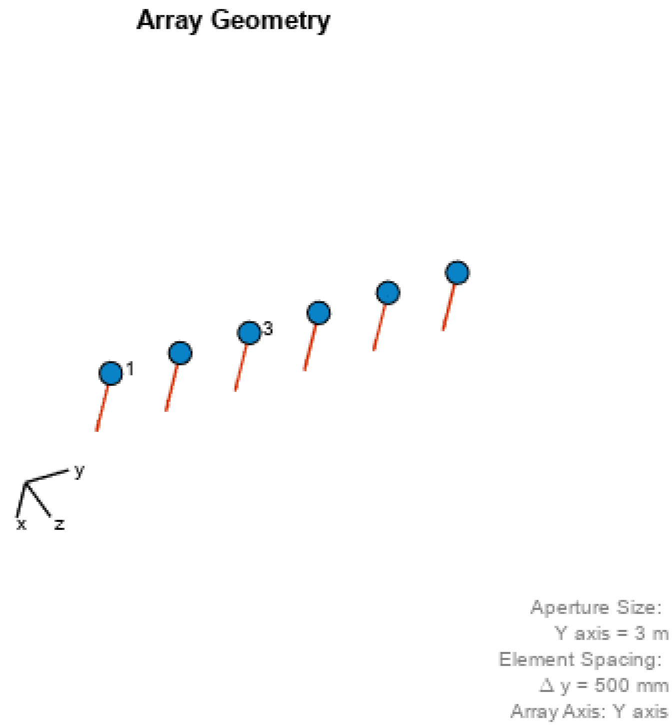
Examples

Geometry and Indices of ULA Elements

Draw a 6-element ULA and use the `'ShowIndex'` parameter to show the indices of the first and third elements.

```
array = phased.ULA(6);
viewArray(array,'ShowIndex',[1 3],'ShowNormals',true, ...
```

```
    'ShowLocalCoordinates',true,'Orientation',[60;100;45], ...
    'ShowAnnotation',true)
```



See Also

`phased.ArrayResponse`

Topics

Phased Array Gallery

phased.UnderwaterRadiatedNoise

Package: `phased`

Radiate acoustic noise from underwater or surface sound source

Description

The phased.UnderwaterRadiatedNoise System object creates a source of underwater radiated acoustic noise. The noise source can either be on the sea surface or underwater. The radiated noise consists of two components: broadband noise and tonal noise. Broadband noise fills the entire operating system bandwidth while tonal noise occurs at discrete frequencies within the bandwidth. In general, the intensity of the radiated noise depends on the noise spectrum and the source radiation pattern. The object lets you specify

- The spectral shape and levels of the broadband noise.
- The frequencies and levels of the tones.
- The noise source radiation pattern.

To propagate noise from a source to a receiver, use this object with the `phased.IsoSpeedUnderwaterPaths` and the `phased.MultipathChannel` objects.

To generate radiated underwater noise:

1. Create the `phased.UnderwaterRadiatedNoise` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
noiseradiator = phased.UnderwaterRadiatedNoise
noiseradiator = phased.UnderwaterRadiatedNoise(Name,Value)
```

Description

`noiseradiator = phased.UnderwaterRadiatedNoise` creates an underwater radiated noise source with default property values.

`noiseradiator = phased.UnderwaterRadiatedNoise(Name,Value)` creates an underwater radiated noise source with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `noiseradiator = phased.UnderwaterRadiatedNoise('TonalLevels',[4700 4900 5150],'SampleRate',500,'OperatingFrequency',5000)` creates a noise source with tones at 4.7, 4.9, and 5.15 kHz. The sample rate is set to 0.5 kHz and the operating frequency is 5 kHz. The broadband noise levels are set to default values.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`NumSamples` — Number of output noise samples

`100` (default) | positive integer

Number of output noise samples, specified as a positive integer.

Example: `500`

Data Types: `double`

`SampleRate` — Sample rate

`1.0e3` (default) | positive scalar

Sample rate, specified as a positive scalar. The sample rate together with the operating frequency determines the operating frequency band. See "Input and Output Frequency Bands" on page 1-1886 for the definition of the operating frequency band. Units are in Hz.

Example: `2.0e3`

Data Types: `double`

`OperatingFrequency` — Signal operating frequency

`20.0e3` (default) | positive scalar

Signal operating frequency, specified as a positive scalar. The operating frequency determines the center of the operating frequency band. See "Input and Output Frequency Bands" on page 1-1886 for the definition of the operating frequency band. Units are in Hz.

Example: `15.0e3`

Data Types: `double`

`TonalFrequencies` — Radiated tonal noise frequencies

`[19700 20100 20300]` (default) | real-valued vector of nonnegative values

Radiated tonal frequencies, specified as a vector of nonnegative values. Tonal frequencies must lie in the operating frequency band. Tonal frequencies outside this band are ignored. The length of the `TonalFrequencies` vector must match the length of the `TonalLevels` vector. Units are in Hz. See "Input and Output Frequency Bands" on page 1-1886 for the definition of the operating frequency band.

Example: `[14900 15010 15200]`

Data Types: `double`

`TonalLevels` — Radiated tonal noise levels

`[150 150 150]` (default) | real-valued vector

Radiated tonal noise levels, specified as a vector of positive values. Units are in dB//1µPa. The length of the `TonalLevels` vector must match the length of the `TonalFrequencies` vector.

Example: `[50 20 170]`

Data Types: `double`

`BroadbandLevel` — Broadband noise spectrum level
`130` (default) | vector of real values

Broadband noise spectrum level, specified as a vector of real-values. This vector specifies the noise spectrum at uniformly spaced frequencies in the operating system band. Units are in dB/Hz//1µPa.

Example: `[140 145 145 130]`

Data Types: `double`

`AzimuthAngles` — Elevation angles of source radiation pattern entries
`-180:180` (default) | vector of real values

Azimuth angles of source radiation pattern entries, specified as a length-*P* vector. This property specifies the azimuth angles of the columns of the source radiation pattern, DirectionalPattern property. *P* must be greater than 2. Units are in degrees.

Example: `[140 145 145 130]`

Data Types: `double`

`ElevationAngles` — Elevation angles of directional radiation pattern
`-90:90` (default) | length-*Q* vector of real values

Elevation angles of the source radiation pattern entries, specified as a length-*Q* vector. This property specifies the elevation angles of the rows of the source radiation pattern, DirectionalPattern . *Q* must be greater than 2. Units are in degrees.

Example: `[-45 -30 0 45 30]`

Data Types: `double`

`DirectionalPattern` — Source radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*K* array | 1-by-*P*-by-*K* array | real-valued *K*-by-*P* matrix

Source radiation pattern, specified as a real-valued matrix or array. Units are in dB. The allowable pattern dimensions are:

Radiation Pattern Dimensions

Dimensions	Application
Q-by-*P* matrix	Specifies a directional pattern as a function of *Q* elevation angles and *P* azimuth angles. The same pattern is used for all frequencies.
Q-by-*P*-by-*K* array	Specifies a directional pattern as a function of *Q* elevation angles, *P* azimuth angles, and *K* frequencies. If $K = 1$, the directional pattern is equivalent to a *Q*-by-*P* matrix.
1-by-*P*-by-*K* array	Specifies a directional pattern as a function of *P* azimuth angles and *K* frequencies. These dimensions apply when there is only one elevation angle.
K-by-*P* matrix	Specifies a directional pattern as a function of *P* azimuth angles and *K* frequencies. These dimensions apply when there is only one elevation angle.

- *Q* is the length of the vector specified by the ElevationAngles property.
- *P* is the length of the vector specified by the AzimuthAngles property.
- *K* is the number of frequencies specified by the FrequencyVector property.

Matrix and Array Specifications

Application	Radiation Pattern Dimensions
One source and *M* radiation directions specified in the `ang` argument of the object function.	Specify one radiation pattern matrix or array for all radiating angles.
M sources with the same pattern and *M* radiation directions specified in the `ang` argument of the object function.	Specify one radiation pattern matrix or array for all radiating angles.
M sources with individual radiation patterns and *M* radiation directions specified in the `ang` argument of the object function.	*M* radiation patterns in a cell array. All patterns must have the same sizes and types. The number of patterns must match the number of radiating angles.

Example: `[1,3;5,-10]`

Data Types: `double`

`FrequencyVector` — Radiation pattern frequencies
`[0 100e6]` (default) | positive, real-valued 1-by-*K* vector

Radiation pattern frequencies, specified as a positive, real-valued 1-by-*K* vector. The vector defines the frequencies at which the DirectionalPattern property values are specified. The elements of the vector must be in strictly increasing order and frequencies must lie in the operating frequency band. See “Input and Output Frequency Bands” on page 1-1886 for the definition of the operating frequency band. Units are in Hz.

Example: `1e6`

Data Types: `double`

`SeedSource` — Random number generator seed source
`'Auto'` (default) | `'Property'`

Random number generator seed source, specified as `'Auto'` or `'Property'`. The random numbers are used to generate the noise. When you set this property to `'Auto'`, random numbers are generated using the default MATLAB random number generator. When you set this property to `'Property'`, the object uses a private random number generator with a seed specified by the `Seed` property.

To use this object with Parallel Computing Toolbox software, set this property to `'Auto'`.

Data Types: `char`

Seed — Random number generator seed
0 (default) | nonnegative integer less than 2^{32}.

Random number generator seed, specified as a nonnegative integer less than 2^{32}.

Example: `10223`

Dependencies

To enable this property, set the SeedSource property to `'Property'`.

Data Types: `double`

Usage

Syntax

`y = radiatednoise(ang)`

Description

`y = radiatednoise(ang)` returns the noise,`y`, radiated in the direction, `ang`.

Input Arguments

ang — Noise radiation directions
real-valued 2-by-*M* matrix

Noise radiation directions, specified as a real-valued 2-by-*M* matrix. Each column of `ang` specifies the direction of radiation of the corresponding noise signal in the form `[AzimuthAngle;ElevationAngle]`. When `ang` represents multiple angles, the DirectionalPattern property can contain one pattern or *M* patterns. In that case, each column of `ang` corresponds to one of the patterns. If there is only one pattern, then the multiple noise signals are generated using the same source pattern. Units are in degrees.

Example: `[0 20; 35 -10]`

Data Types: `double`

Output Arguments

y — Radiated noise
complex-valued *M*-by-*N* matrix

Radiated noise, specified as a complex-valued *M*-by-*N* matrix. *M* is the number of angles specified in the `ang` argument. *N* is the number of samples specified by the `NumSamples` property. Radiated noise lies in the baseband range *[-fs/2 fs/2]*. f_s represents the sample rate set by the SampleRate property. Noise units are in Pa.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

The `reset` object function resets the random number generator state when the SeedSource property is set to `'Property'`.

Examples

Radiate Underwater Noise from Surface Ship

Generate radiated noise from a surface ship. The sonar operating frequency is 5.0 kHz and the sampling rate is 1.0 kHz. By definition, broadband noise band lies in the band 4.5 kHz to 5.5 kHz. In addition, there are tonal noises at 4.6, 5.2, and 5.4 kHz.

```
shippos = [0;0;0];
rcvpos = [100;0;-50];
```

Compute the noise transmission angle from the ship to the receiver.

```
[~,ang] = rangeangle(rcvpos,shippos)
```

```
ang = 2×1

         0
  -26.5651
```

Construct a `phased.UnderwaterRadiatedNoise` System object™ having a radiation pattern that depends only on elevation angle. Compute the noise radiated in the direction of the receiver. Create 10000 samples of the noise radiated towards the target.

```
azang = [-180:180];
elang = [-80:80];
pattern = mag2db(repmat(cosd(elang)',1,numel(azang)));
fs = 1000;
noiseradiator = phased.UnderwaterRadiatedNoise('NumSamples',10000, ...
    'SampleRate',fs,'TonalFrequencies',[4600 5200 5400],'TonalLevels',[200,200,200], ...
    'BroadbandLevels',[180 180 190 190 190 188 185],'AzimuthAngles',azang, ...
```

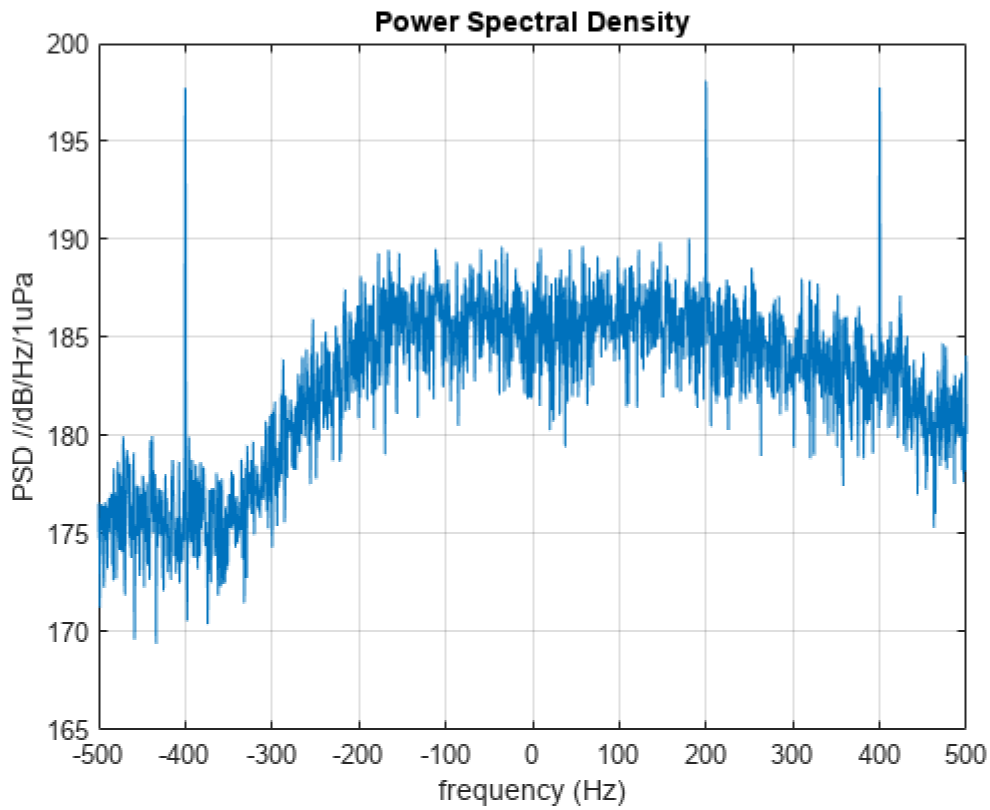
```
    'ElevationAngles',elang,'DirectionalPattern',pattern, ...
    'OperatingFrequency',5e3,'SeedSource','Property','Seed',2781);
```

Generate 10000 samples of noise.

```
y = noiseradiator(ang);
```

Plot the noise power spectral density (psd). Convert the psd to intensity referenced to 1uPa.

```
[psd,fr] = pwelch(y,[],[],[],noiseradiator.SampleRate,'psd','centered');
plot(fr,10*log10(psd*1e12));
title('Power Spectral Density')
xlabel('frequency (Hz)')
ylabel('PSD //dB/Hz/1uPa')
grid
```



The three tones appear over the broadband spectrum.

Radiate Underwater Noise with Frequency-Dependent Pattern

Generate radiated noise from an underwater vehicle. Assume that the noise radiation pattern depends on frequency. The sonar operating frequency is 5.0 kHz and the sampling rate is 1.0 kHz. By definition, the broadband noise band is from 4.5 kHz to 5.5 kHz. In addition, there are tonal noises at 4.6, 5.2, and 5.3 kHz. Define the radiation pattern at three frequencies within this band. All three patterns are multiples of the basic pattern. The frequencies of the radiation patterns are 4.6 kHz, 5.0 kHz, and 5.3 kHz.

First, specify the source and receiver positions.

```
srcpos = [0;50;-20];
rcvpos = [100;0;-50];
```

Compute the noise transmission angle from vehicle to receiver.

```
[~,ang] = rangeangle(rcvpos,srcpos)
```

```
ang = 2×1

  -26.5651
  -15.0203
```

Construct a `phased.UnderwaterRadiatedNoise` System object™ with a radiation pattern that depends only on the azimuth angle and frequency. Compute the noise radiated in the direction of the receiver. Create 10000 samples of noise radiated from the vehicle.

```
azang = [-180:180];
elang = [-90:90];
fc = 5000.0;
```

Put the radiation pattern in a three-dimensional array.

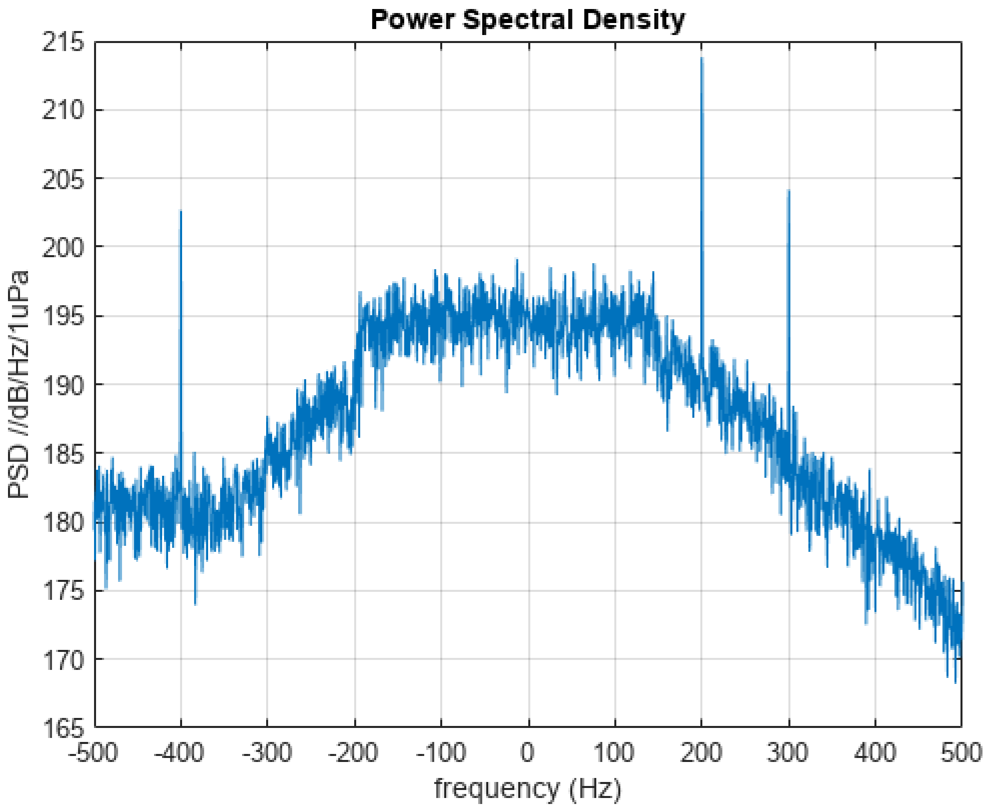
```
basepattern = repmat(10*cosd(azang).^2,numel(elang),1);
pattern(:,:,1) = 0.5*basepattern;
pattern(:,:,2) = basepattern;
pattern(:,:,3) = 0.6*basepattern;
patterndb = mag2db(pattern);
noiseradiator = phased.UnderwaterRadiatedNoise('NumSamples',10000, ...
    'SampleRate',1e3,'TonalFrequencies',[4600,5200 5300], ...
    'TonalLevels',[200,210,200],'BroadbandLevels',[180 180 190 190 190 180 170], ...
    'AzimuthAngles',azang,'ElevationAngles',elang, ...
    'FrequencyVector',[4600,5000,5300],'DirectionalPattern',pattern, ...
    'OperatingFrequency',5e3,'SeedSource','Property','Seed',2081);
```

Generate 10000 samples of noise.

```
y = noiseradiator(ang);
```

Plot the noise power spectral density (psd). Convert the psd to intensity referenced to 1uPa.

```
[psd,fr] = pwelch(y,[],[],[],noiseradiator.SampleRate,'psd','centered');
plot(fr,10*log10(psd*1e12));
title('Power Spectral Density')
xlabel('frequency (Hz)')
ylabel('PSD //dB/Hz/1uPa')
grid
```



The three tones appear over the broadband spectrum.

Radiate Underwater Noise from Two Sources

Generate radiated noise from a two underwater vehicles. Assume that the noise radiation pattern is different for each. The sonar operating frequency is 5.0 kHz and the sampling rate is 1.0 kHz. By definition, the broadband noise band is from 4.5 kHz to 5.5 kHz. In addition, there are tonal noises at 4.6, 5.2, and 5.3 kHz. The frequencies of the radiation patterns are 4.6 kHz, 5.0 kHz, and 5.3 kHz.

First, specify the source and receiver positions.

```
srcpos1 = [0;50;-20];
srcpos2 = [200;50;-80];
rcvpos = [100;0;-50];
```

Compute the noise transmission angle from vehicle to receiver.

```
[~,ang1] = rangeangle(rcvpos,srcpos1);
[~,ang2] = rangeangle(rcvpos,srcpos2);
```

Construct a `phased.UnderwaterRadiatedNoise` System object™ with a radiation pattern that depends only on the azimuth angle and frequency. Compute the noise radiated in the direction of the receiver. Create 10000 samples of noise radiated from the vehicle.

```
azang = [-180:180];
elang = [-90:90];
fc = 5000.0;
```

Put the radiation pattern in a three-dimensional array.

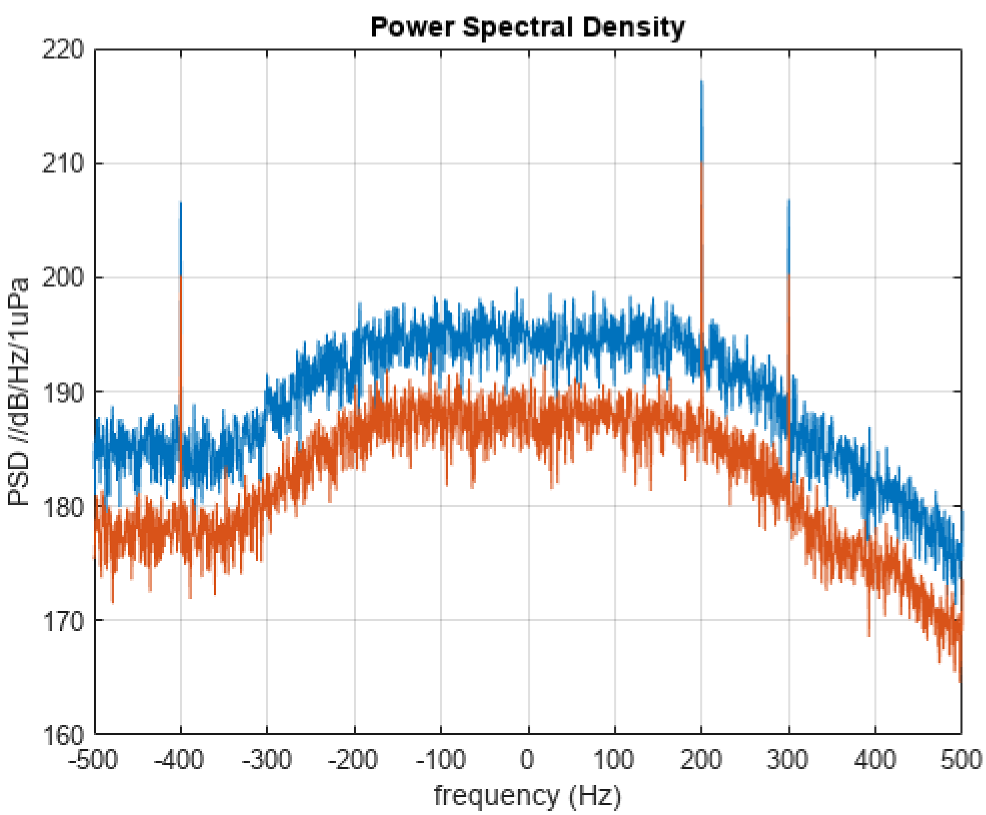
```
pattern1 = repmat(10*cosd(azang).^2,numel(elang),1);
pattern2 = ones(181,361);
pattern1db = mag2db(pattern1);
pattern2db = mag2db(pattern2);
noiseradiator = phased.UnderwaterRadiatedNoise('NumSamples',10000, ...
    'SampleRate',1e3,'TonalFrequencies',[4600,5200 5300], ...
    'TonalLevels',[200,210,200],'BroadbandLevels',[180 180 190 190 190 180 170], ...
    'AzimuthAngles',azang,'ElevationAngles',elang, ...
    'FrequencyVector',[4600,5000,5300],'DirectionalPattern',{pattern1,pattern2}, ...
    'OperatingFrequency',5e3,'SeedSource','Property','Seed',2081);
```

Generate 10000 samples of noise.

```
y = noiseradiator([ang1,ang2]);
```

Plot the noise power spectral density (psd). Convert the psd to intensity referenced to 1uPa.

```
[psd,fr] = pwelch(y,[],[],[],noiseradiator.SampleRate,'psd','centered');
plot(fr,10*log10(psd*1e12));
title('Power Spectral Density')
xlabel('frequency (Hz)')
ylabel('PSD //dB/Hz/1uPa')
grid
```



The three tones appear over the broadband spectrum.

More About

Input and Output Frequency Bands

The specified broadband and tonal noise frequencies must lie inside the operating frequency band,

$[f_c - f_s/2, f_c + f_s/2]$

f_s represents the sample rate set by the SampleRate property and f_c represents the operating frequency, set by the OperatingFrequency property.

However, the output noise spectrum lies in baseband:

$[- f_s/2, f_s/2]$

Version History

Introduced in R2017b

References

[1] Urick, R.J. *Principles of Underwater Sound, 3rd Edition*. New York: Peninsula Publishing, 1996.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Objects

`phased.BackscatterSonarTarget` | `phased.IsoSpeedUnderwaterPaths` | `phased.IsotropicHydrophone` | `phased.IsotropicProjector` | `phased.MultipathChannel`

Functions

`range2tl` | `sonareqsl` | `sonareqsnr` | `sonareqtl` | `tl2range`

phased.URA

Package: phased

Uniform rectangular array

Description

To create a Uniform Rectangular Array (URA) System object:

1. Create the phased.URA object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
array = phased.URA
array = phased.URA(Name,Value)
array = phased.URA(SZ,D,Name,Value)
```

Description

array = phased.URA creates a uniform rectangular array (URA) System object that models a URA formed from identical isotropic phased array elements. Array elements are contained in the *yz*-plane in a rectangular lattice. The array look direction (boresight) points along the positive *x*-axis.

array = phased.URA(Name,Value) creates an array object with each specified property Name set to the specified Value. You can specify additional name-value pair arguments in any order as (Name1,Value1, ..., NameN,ValueN). All properties needed to fully specify this object can be found in Properties."Response of 2-by-2 URA of Short-Dipole Antennas" on page 1-1891

array = phased.URA(SZ,D,Name,Value) creates a phased.URA array System object with its Size property set to SZ and its ElementSpacing property set to D. Other specified property Names are set to the specified Values. SZ and D are value-only arguments. When specifying a value-only argument, specify all preceding value-only arguments. You can specify name-value pair arguments in any order.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the release function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

Element — phased array element
isotropic antenna element System object with default properties (default) | Phased Array System Toolbox antenna, microphone, or transducer element System object | Antenna Toolbox antenna System object

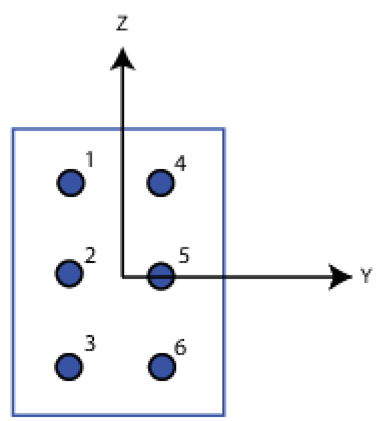
Phased array element, specified as a Phased Array System Toolbox antenna, microphone, or transducer element or Antenna Toolbox antenna.

Example: `phased.CosineAntennaElement`

Size — Array size
`[2 2]` (default) | positive scalar | 1-by-2 vector of positive values

Array size, specified as a 1-by-2 vector of integers or a single integer. containing the size of the array. If `Size` is a 1-by-2 vector, the vector has the form `[NumberOfRows, NumberOfColumns]`. If `Size` is a scalar, the array has the same number of elements in each row and column. For a URA, array elements are indexed from top to bottom along a column and continuing to the next columns from left to right. In this illustration, a `Size` value of `[3,2]` array has three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Example: `[3,2]`

Data Types: `double`

ElementSpacing — Element spacing
`[0.5 0.5]` (default) | positive scalar | 1-by-2 vector of positive values

Element spacing, specified as a positive scalar or 1-by-2 vector of positive values. If `ElementSpacing` is a 1-by-2 vector, it has the form `[SpacingBetweenRows,SpacingBetweenColumns]`. See “Spacing Between Columns” on page 1-1907 and “Spacing Between Rows” on page 1-1907. If `ElementSpacing` is a scalar, both row and column spacings are equal. Units are in meters.

Example: `[0.3, 0.5]`

Data Types: `double`

Lattice — Lattice type
'Rectangular' (default) | 'Triangular'

Element lattice type, specified as 'Rectangular' or 'Triangular'. When you set the Lattice property to 'Rectangular', all elements of the URA are aligned in both row and column directions. When you set the Lattice property to 'Triangular', elements in even rows are displaced toward the positive row axis direction. The displacement is one-half the element spacing along the row.

Example: 'Triangular'

Data Types: char | string

ArrayNormal — Array normal direction
'x' (default) | 'y' | 'z'

Array normal direction, specified as one of 'x', 'y', or 'z'. URA elements lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

ArrayNormal Property Value	Element Positions and Boresight Directions
'x'	Array elements lie on the *yz*-plane. All element boresight vectors point along the *x*-axis. This is the default value.
'y'	Array elements lie on the *zx*-plane. All element boresight vectors point along the *y*-axis.
'z'	Array elements lie on the *xy*-plane. All element boresight vectors point along the *z*-axis.

Taper — Element tapers
1 (default) | complex-valued scalar | complex-valued 1-by-*MN* row vector

Element tapers, specified as a complex-valued scalar, complex-valued 1-by-*MN* vector, or complex-valued *M*-by-*N* matrix. Tapers are applied to each element in the sensor array. Tapers are often referred to as element weights. *M* is the number of elements along the *z*-axis, and *N* is the number of elements along *y*-axis. *M* and *N* correspond to the values of [NumberofRows,NumberOfColumns] in the SIze property. If Taper is a scalar, the same taper value is applied to all elements. If the value of Taper is a vector or matrix, taper values are applied to the corresponding elements. Tapers are used to modify both the amplitude and phase of the received data.

Example: [0.4 1 0.4]

Data Types: double

Usage

Syntax

```
RESP = array(FREQ,ANG)
```

Description

RESP = array(FREQ,ANG) returns the responses of the array elements, RESP, at the operating frequencies specified in FREQ and directions specified in ANG.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

FREQ — Array operating frequencies
length-*L* row vector of positive values

Array operating frequencies, specified as a length-*L* row vector. Typical values are within the range specified by a frequency range property of the array element. That property is called `FrequencyRange` or `FrequencyVector`, depending on the type of array element. The element has zero response at frequencies outside that range. Units are in Hz.

Data Types: `double`

ANG — Array response directions
length-*M* row vector with real values | 2-by-*M* matrix with real values

Array response directions, specified as a 2-by-*M* matrix or length-*M* row vector.

If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

If `ang` is a row vector of length *M*, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be 0°.

Units are in degrees.

Output Arguments

RESP — Voltage responses of phased array elements
N-by-*M*-by- complex-valued MATLAB array | MATLAB `struct`

Voltage responses of phased array elements. The output depends on whether or not the array supports polarization.

- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array. The dimension *M* is the number of angles specified in `ANG`. *L* is the number of frequencies specified in `FREQ`. For any element, the columns of `RESP` contain the responses of the array elements for the corresponding direction specified in `ang`. Each of the *L* pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `freq`.
- If the array is capable of supporting polarization, the voltage response `RESP` is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array, and *M* is the number of angles specified in `ANG`. *L* is the number of frequencies specified in `FREQ`. Each column of `RESP` contains the responses of the array elements for the corresponding direction specified in `ANG`. Each of the *L* pages of `ANG` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Specific to phased.URA and other array System objects

beamwidth	Compute and display beamwidth of an array
collectPlaneWave	Simulate received plane waves at array
directivity	Compute array directivity
getElementNormal	Normal vectors for array elements
getElementPosition	Positions of array elements
getNumElements	Number of elements in an array
getTaper	Array element tapers
isPolarizationCapable	Array polarization capability
pattern	Plot array directivity and patterns
patternAzimuth	Plot array directivity or pattern versus azimuth
patternElevation	Plot array directivity or pattern versus elevation
perturbations	Perturbations defined on array
perturbedArray	Apply perturbations to phased array
perturbedPattern	Display pattern of perturbed array
plotGratingLobeDiagram	Plot grating lobe diagram of array
viewArray	View array geometry

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Response of 2-by-2 URA of Short-Dipole Antennas

Construct a 2-by-2 rectangular lattice URA of short-dipole antenna elements. Then, find the response of each element at boresight. Assume the operating frequency is 1 GHz.

```
sSD = phased.ShortDipoleAntennaElement;
sURA = phased.URA('Element',sSD,'Size',[2 2]);
fc = 1e9;
ang = [0;0];
resp = step(sURA,fc,ang);
disp(resp.V)
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

Azimuth Response of a 3-by-2 URA at Boresight

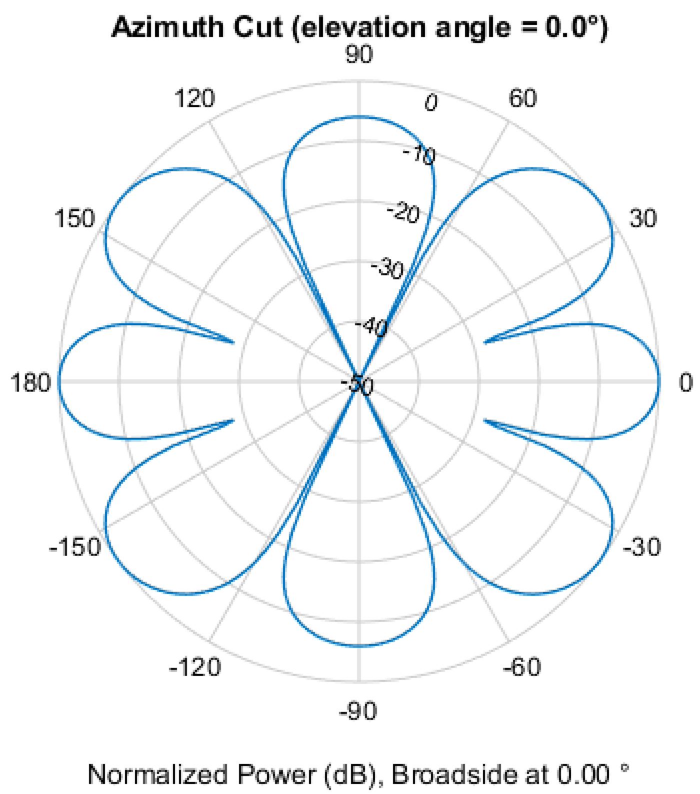
Construct a 3-by-2 rectangular lattice URA. By default, the array consists of isotropic antenna elements. Find the response of each element at boresight, 0 degrees azimuth and elevation. Assume the operating frequency is 1 GHz.

```
array = phased.URA('Size',[3 2]);
fc = 1e9;
ang = [0;0];
resp = array(fc,ang);
disp(resp)
```

```
     1
     1
     1
     1
     1
     1
```

Plot the azimuth pattern of the array.

```
c = physconst('LightSpeed');
pattern(array,fc,[-180:180],0,'PropagationSpeed',c, ...
    'CoordinateSystem','polar','Type','powerdb','Normalize',true)
```



Compare Triangular and Rectangular Lattice URAs

This example shows how to find and plot the positions of the elements of a 5-row-by-6-column URA with a triangular lattice and a URA with a rectangular lattice. The element spacing is 0.5 meters for both lattices.

Create the arrays.

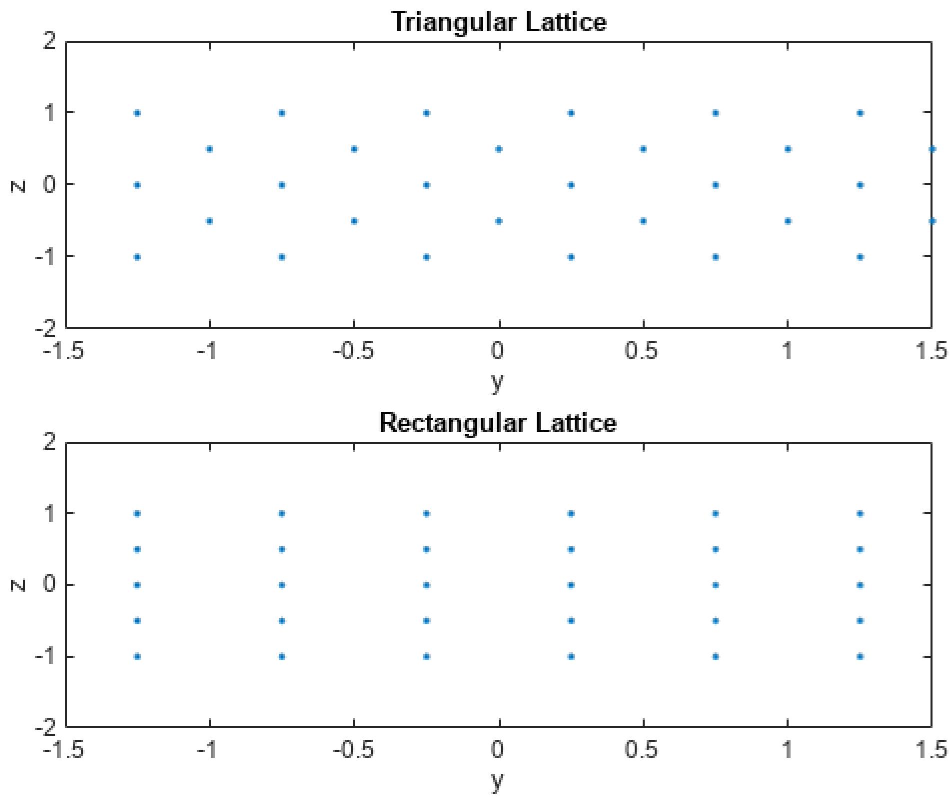
```
h_tri = phased.URA('Size',[5 6],'Lattice','Triangular');
h_rec = phased.URA('Size',[5 6],'Lattice','Rectangular');
```

Get the element y,z positions for each array. All the x coordinates are zero.

```
pos_tri = getElementPosition(h_tri);
pos_rec = getElementPosition(h_rec);
pos_yz_tri = pos_tri(2:3,:);
pos_yz_rec = pos_rec(2:3,:);
```

Plot the element positions in the yz-plane.

```
figure;
gcf.Position = [100 100 300 400];
subplot(2,1,1);
plot(pos_yz_tri(1,:), pos_yz_tri(2,:), '.')
axis([-1.5 1.5 -2 2])
xlabel('y'); ylabel('z')
title('Triangular Lattice')
subplot(2,1,2);
plot(pos_yz_rec(1,:), pos_yz_rec(2,:), '.')
axis([-1.5 1.5 -2 2])
xlabel('y'); ylabel('z')
title('Rectangular Lattice')
```



Adding Tapers to an Array

Construct a 5-by-2 element URA with a Taylor window taper along each column. The tapers form a 5-by-2 matrix.

```
taper = taylorwin(5);
ha = phased.URA([5,2],'Taper',[taper,taper]);
w = getTaper(ha)
```

```
w = 10×1

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

Simulate Received Signal at URA

Simulate two received random signals at a 6-element URA. The array has a rectangular lattice with two elements in the row direction and three elements in the column direction. The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
array = phased.URA([2 3]);
fc = 100e6;
y = collectPlaneWave(array,randn(4,2),[10 30],fc,physconst('LightSpeed'));
```

Directivity of Uniform Rectangular Array

Compute the directivity of two uniform rectangular arrays (URA). The first array consists of isotropic antenna elements. The second array consists of cosine antenna elements. In addition, compute the directivity of the first array steered to a specific direction.

Array of isotropic antenna elements

First, create a 10-by-10-element URA of isotropic antenna elements spaced one-quarter wavelength apart. Set the signal frequency to 800 MHz.

```
c = physconst('LightSpeed');
fc = 3e8;
lambda = c/fc;
myAntIso = phased.IsotropicAntennaElement;
myArray1 = phased.URA;
myArray1.Element = myAntIso;
myArray1.Size = [10,10];
myArray1.ElementSpacing = [lambda*0.25,lambda*0.25];
ang = [0;0];
d = directivity(myArray1,fc,ang,'PropagationSpeed',c)
```

```
d = 15.7753
```

Array of cosine antenna elements

Next, create a 10-by-10-element URA of cosine antenna elements also spaced one-quarter wavelength apart.

```
myAntCos = phased.CosineAntennaElement('CosinePower',[1.8,1.8]);
myArray2 = phased.URA;
myArray2.Element = myAntCos;
myArray2.Size = [10,10];
myArray2.ElementSpacing = [lambda*0.25,lambda*0.25];
ang = [0;0];
d = directivity(myArray2,fc,ang,'PropagationSpeed',c)
```

```
d = 19.7295
```

The directivity is increased due to the directivity of the cosine antenna elements.

Steered array of isotropic antenna elements

Finally, steer the isotropic antenna array to 30 degrees in azimuth and examine the directivity at the steered angle.

```
ang = [30;0];
w = steervec(getElementPosition(myArray1)/lambda,ang);
d = directivity(myArray1,fc,ang,'PropagationSpeed',c,...
    'Weights',w)
```

```
d = 15.3309
```

The directivity is maximum in the steered direction and equals the directivity of the unsteered array at boresight.

URA Element Normals

Construct three 2-by-2 URA's with element normals along the *x*-, *y*-, and *z*-axes. Obtain the element positions and normal directions.

First, choose the array normal along the *x*-axis.

```
sURA1 = phased.URA('Size',[2,2],'ArrayNormal','x');
pos = getElementPosition(sURA1)
```

```
pos = 3×4

         0         0         0         0
   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
```

```
normvec = getElementNormal(sURA1)
```

```
normvec = 2×4

     0     0     0     0
     0     0     0     0
```

All elements lie in the *yz*-plane and the element normal vectors point along the *x*-axis *(0°,0°)*.

Next, choose the array normal along the *y*-axis.

```
sURA2 = phased.URA('Size',[2,2],'ArrayNormal','y');
pos = getElementPosition(sURA2)
```

```
pos = 3×4

    0.2500    0.2500   -0.2500   -0.2500
         0         0         0         0
    0.2500   -0.2500    0.2500   -0.2500
```

```
normvec = getElementNormal(sURA2)
```

```
normvec = 2×4

    90    90    90    90
     0     0     0     0
```

All elements lie in the *zx*-plane and the element normal vectors point along the *y*-axis *(90°,0°)*.

Finally, set the array normal along the *z*-axis. Obtain the normal vectors of the odd-numbered elements.

```
sURA3 = phased.URA('Size',[2,2],'ArrayNormal','z');
pos = getElementPosition(sURA3)
```

```
pos = 3×4

   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
         0         0         0         0
```

```
normvec = getElementNormal(sURA3,[1,3])
```

```
normvec = 2×2

     0     0
    90    90
```

All elements lie in the *xy*-plane and the element normal vectors point along the *z*-axis *(0°,90°)*.

Obtain URA Element Positions

Construct a default URA with a rectangular lattice, and obtain the element positions.

```
array = phased.URA;
pos = getElementPosition(array)
```

```
pos = 3×4

         0         0         0         0
   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
```

Obtain Number of URA Elements

Construct a default URA, and obtain the number of elements.

```
array = phased.URA;
N = getNumElements(array)
```

```
N = 4
```

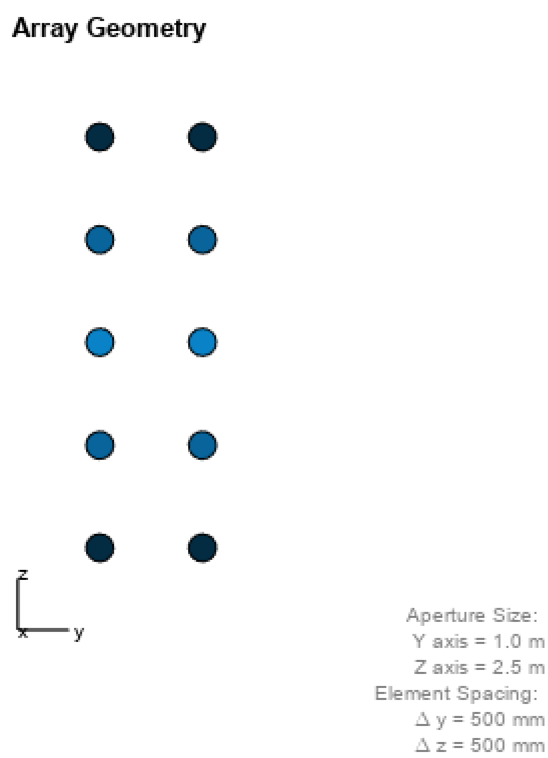
Create Tapered URA

Construct a 5-by-2 element URA with a Taylor window taper along each column. Then, draw the array showing the element taper shading.

```
taper = taylorwin(5);
array = phased.URA([5,2],'Taper',[taper,taper]);
w = getTaper(array)
```

```
w = 10×1

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

```
viewArray(array,'ShowTaper',true)
```



Short-Dipole URA Supports Polarization

Show that a URA array of `phased.ShortDipoleAntennaElement` short-dipole antenna elements supports polarization.

```
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[1e9 10e9]);
array = phased.URA([3,2],'Element',antenna);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

The returned value `1` shows that this array supports polarization.

Pattern of 5x7-Element URA Antenna Array

Create a 5x7-element URA operating at 1 GHz. Assume the elements are spaced one-half wavelength apart. Show the 3-D array patterns.

Create the array

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[50e6,1000e6],...
    'AxisDirection','Z');
fc = 500e6;
c = physconst('LightSpeed');
lam = c/fc;
sURA = phased.URA('Element',sSD,...
    'Size',[5,7],...
    'ElementSpacing',0.5*lam);
```

Call the `step` method

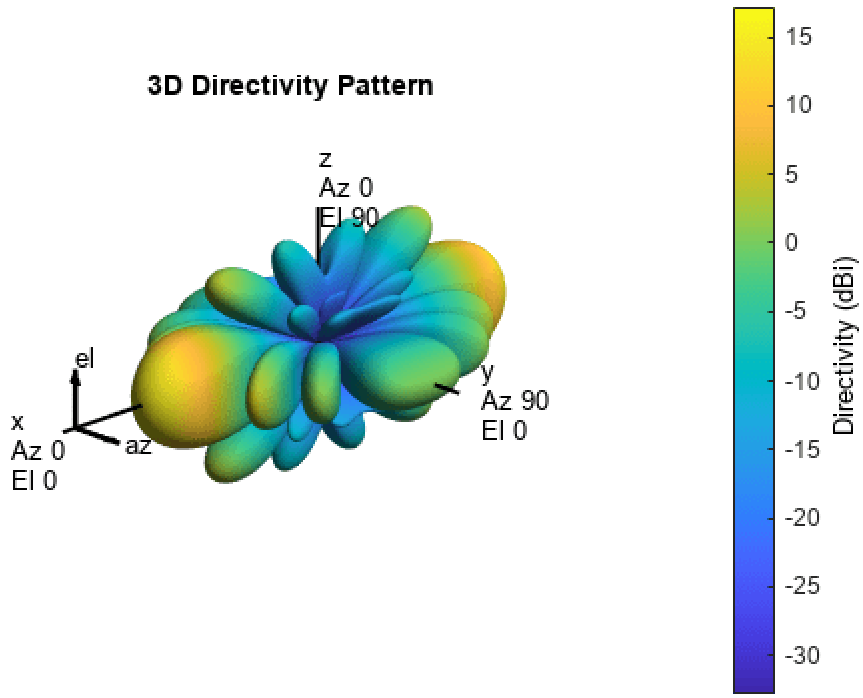
Evaluate the fields of the first five elements at 45 degrees azimuth and 0 degrees elevation.

```
ang = [45;0];
resp = step(sURA,fc,ang);
disp(resp.V(1:5))
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

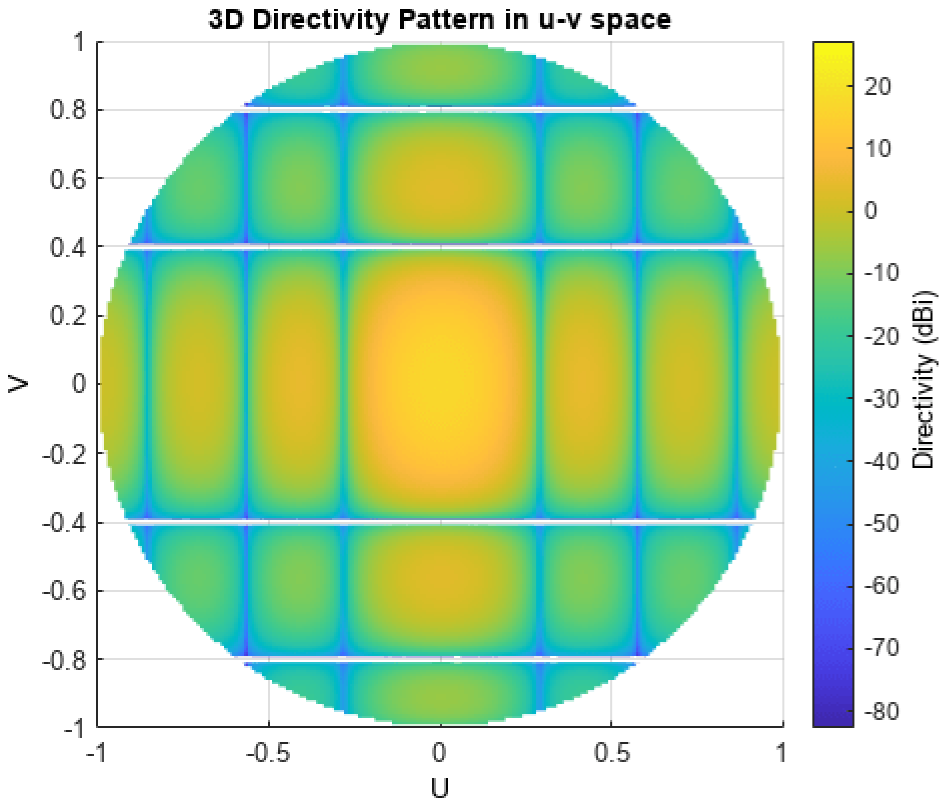
Display the 3-D directivity pattern at 1 GHz in polar coordinates

```
pattern(sURA,fc,[-180:180],[-90:90],...
    'CoordinateSystem','polar',...
    'Type','directivity','PropagationSpeed',c)
```



Display the 3-D directivity pattern at 1 GHz in UV coordinates

```
pattern(sURA,fc,[-1.0:.01:1.0],[-1.0:.01:1.0],...
    'CoordinateSystem','uv',...
    'Type','directivity','PropagationSpeed',c)
```



Azimuth Pattern of 5x7-Element URA Antenna Array

Create a 5x7-element URA of short-dipole antenna elements operating at 1 GHz. Assume the elements are spaced one-half wavelength apart. Plot the array azimuth directivity patterns for two different elevation angles, 0 and 15 degrees. The `patternAzimuth` method always plots the array pattern in polar coordinates.

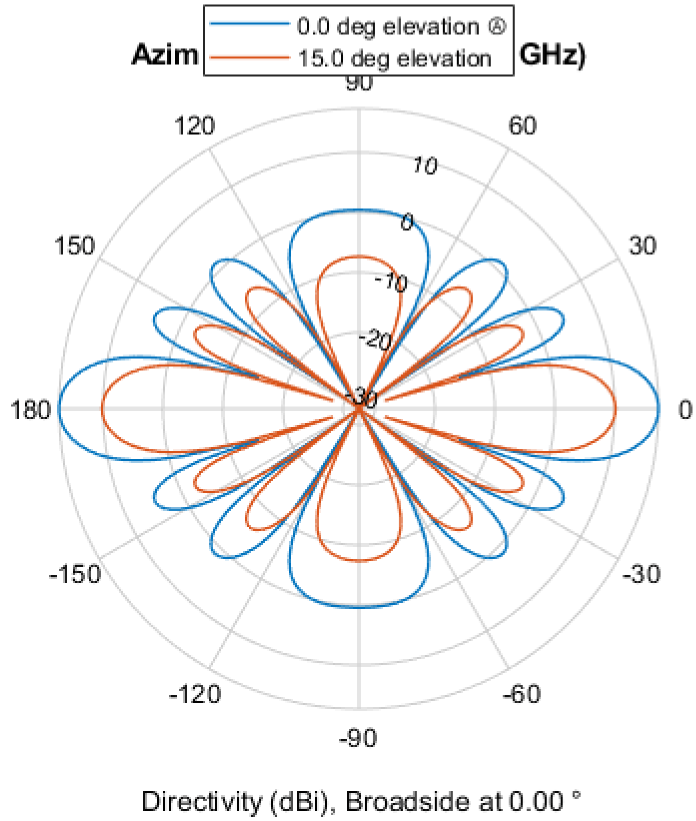
Create the array

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[50e6,1000e6],...
    'AxisDirection','Z');
fc = 1e9;
c = physconst('LightSpeed');
lam = c/fc;
sURA = phased.URA('Element',sSD,...
    'Size',[5,7],...
    'ElementSpacing',0.5*lam);
```

Display the pattern

Display the azimuth directivity pattern at 1 GHz in polar coordinates

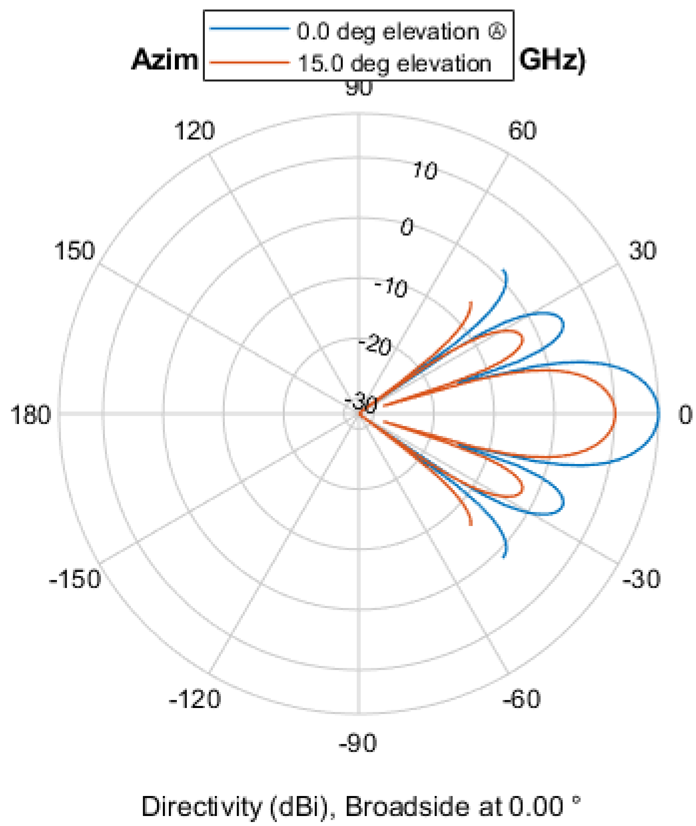
```
patternAzimuth(sURA,fc,[0 15],...
    'PropagationSpeed',c,...
    'Type','directivity')
```



Display a subset of angles

You can plot a smaller range of azimuth angles by setting the `Azimuth` parameter.

```
patternAzimuth(sURA,fc,[0 15],...
    'PropagationSpeed',c,...
    'Type','directivity',...
    'Azimuth',[-45:45])
```



Elevation Pattern of 7x7-Element Acoustic URA

Create a 7x7-element URA of backbaffled omnidirectional transducer elements operating at 2 kHz. Assume the speed of sound in water is 1500 m/s. The elements are spaced less than one-half wavelength apart. Plot the array elevation directivity patterns for three different azimuth angles, -20, 0, and 15 degrees. The `patternElevation` method always plots the array pattern in polar coordinates.

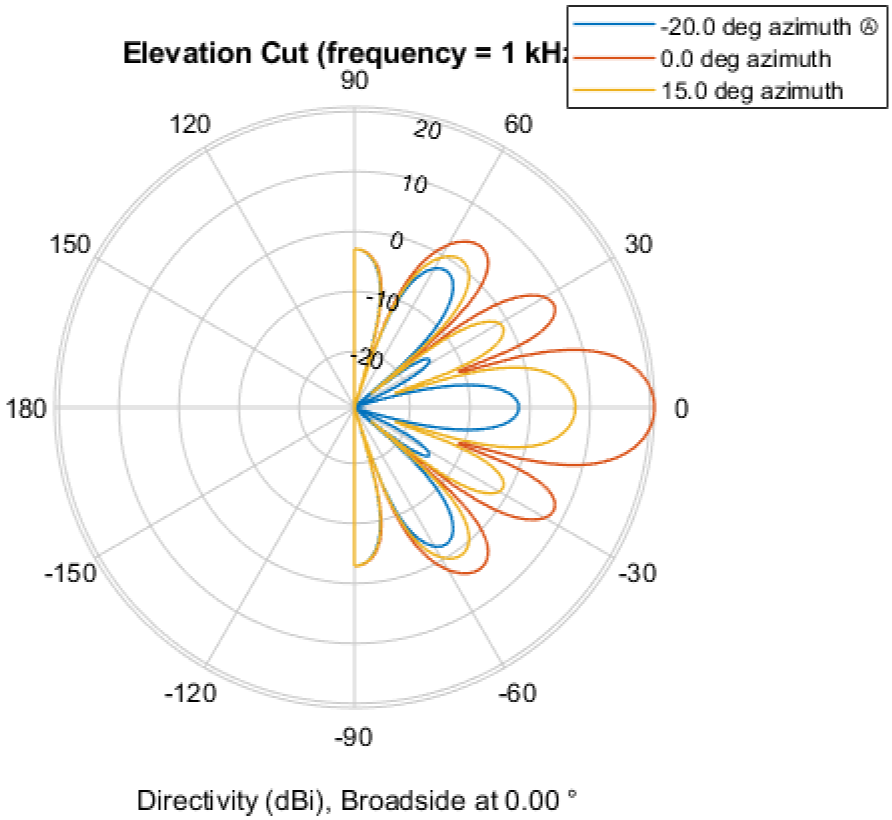
Create the array

```
element = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20,3000],...
    'BackBaffled',true);
fc = 1000;
c = 1500;
lam = c/fc;
array = phased.URA('Element',element,...
    'Size',[7,7],...
    'ElementSpacing',0.45*lam);
```

Display the pattern

Display the azimuth directivity pattern at 1 GHz in polar coordinates.

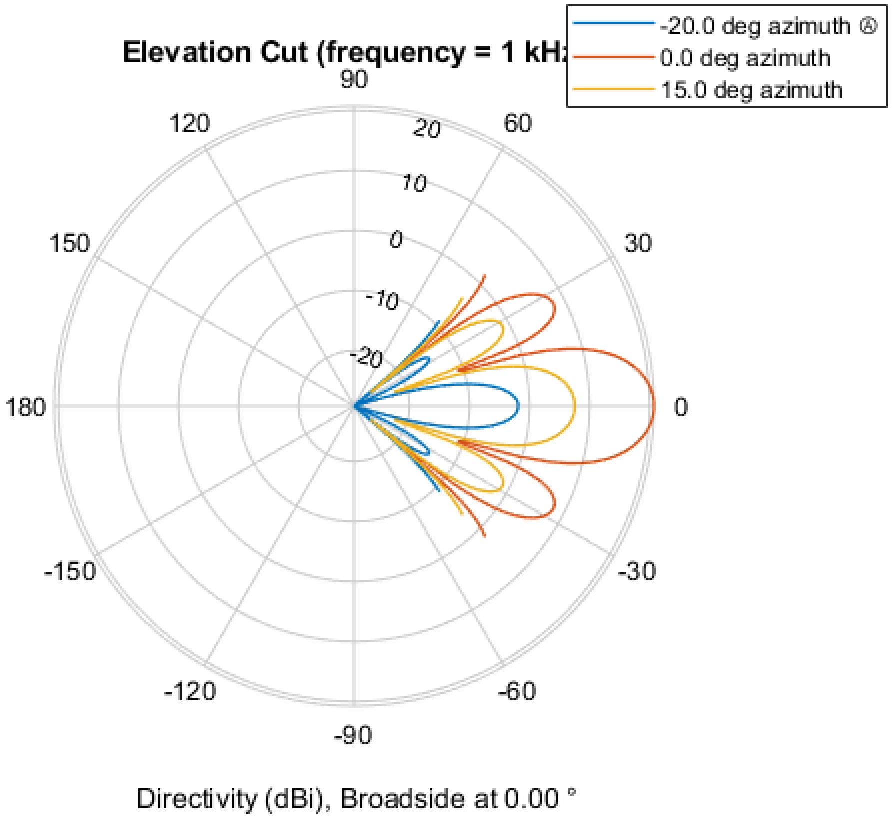
```
patternElevation(array,fc,[-20, 0, 15],...
    'PropagationSpeed',c,...
    'Type','directivity')
```



Display a subset of elevation angles

You can plot a smaller range of elevation angles by setting the `Elevation` parameter.

```
patternElevation(array,fc,[-20, 0, 15],...
    'PropagationSpeed',c,...
    'Type','directivity',...
    'Elevation',[-45:45])
```

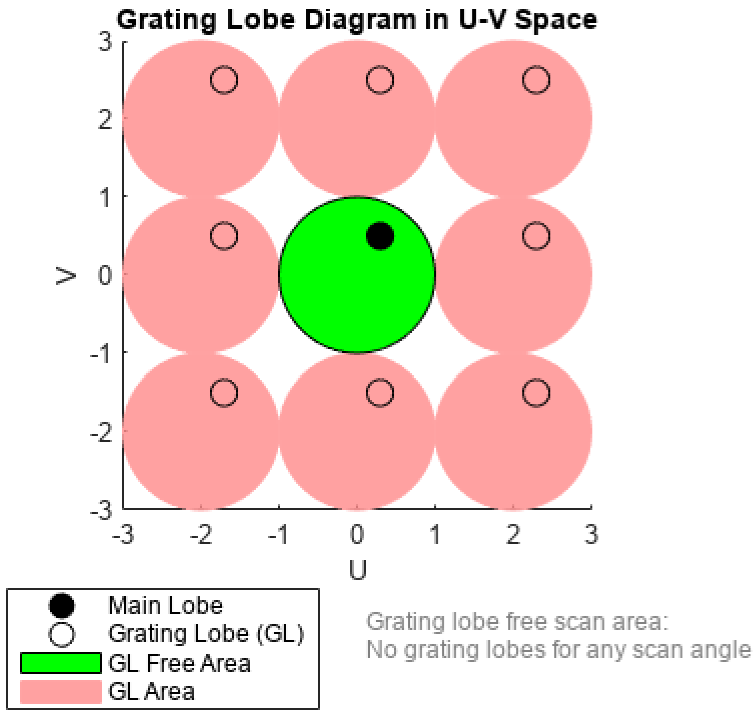


Grating Lobe Diagram for Microphone URA

Plot the grating lobe diagram for an 11-by-9-element uniform rectangular array having element spacing equal to one-half wavelength.

Assume the operating frequency of the array is 10 kHz. All elements are omnidirectional microphone elements. Steer the array in the direction 20 degrees in azimuth and 30 degrees in elevation. The speed of sound in air is 344.21 m/s at 21 deg C.

```
cair = 344.21;
f = 10.0e3;
lambda = cair/f;
microphone = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
array = phased.URA('Element',microphone,'Size',[11,9],...
    'ElementSpacing',0.5*lambda*[1,1]);
plotGratingLobeDiagram(array,f,[20;30],cair);
```



Plot the grating lobes. The main lobe of the array is indicated by a filled black circle. The grating lobes in visible and nonvisible regions are indicated by unfilled black circles. The visible region is the region in u-v coordinates for which $u^2 + v^2 \leq 1$. The visible region is shown as a unit circle centered at the origin. Because the array spacing is less than one-half wavelength, there are no grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

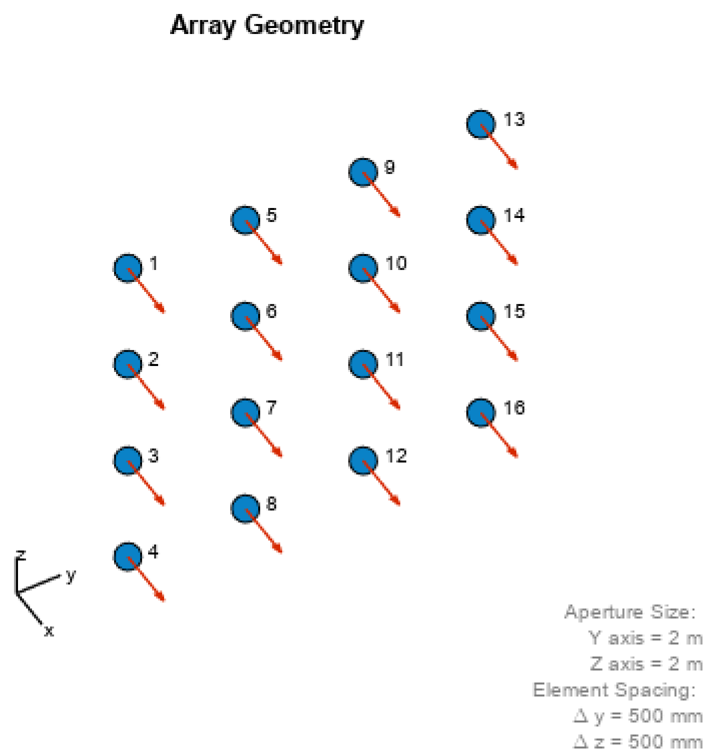
The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it coincides with the visible region.

The white areas of the diagram indicate a region where no grating lobes are possible.

Geometry, Normal Directions, and Indices of URA Elements

This example shows how to display the element positions, normal directions, and indices for all elements of a 4-by-4 square URA.

```
ha = phased.URA(4);
viewArray(ha,'ShowNormals',true,'ShowIndex','All');
```



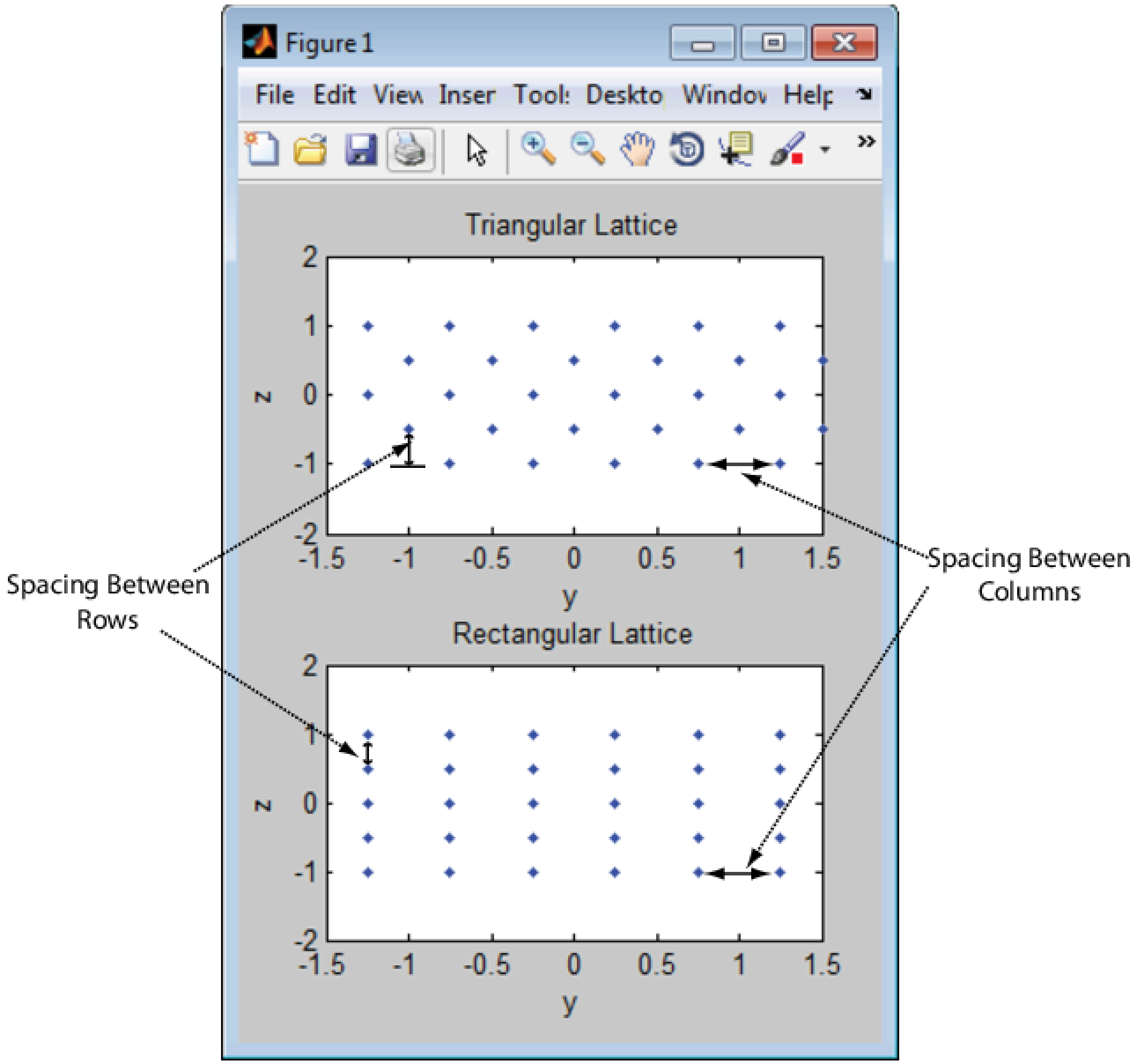
More About

Spacing Between Columns

The spacing between columns is the distance between adjacent elements in the same row.

Spacing Between Rows

The spacing between rows is the distance along the column axis direction between adjacent rows.



Version History

Introduced in R2011a

References

[1] Brookner, E., ed. *Radar Technology*. Lexington, MA: LexBook, 1996.

[2] Brookner, E., ed. *Practical Phased Array Antenna Systems*. Boston: Artech House, 1991.

[3] Mailloux, R. J. “Phased Array Theory and Technology,” *Proceedings of the IEEE*, Vol., 70, Number 3s, pp. 246–291.

[4] Mott, H. *Antennas for Radar and Communications, A Polarimetric Approach*. New York: John Wiley & Sons, 1992.

[5] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- `pattern`, `patternAzimuth`, `patternElevation`, and `viewArray` object functions are not supported.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.HeterogeneousULA` | `phased.HeterogeneousURA` | `uv2azel` | `phitheta2azel`

Topics

Phased Array Gallery

directivity

System object: phased.URA
Package: phased

Directivity of uniform rectangular array

Syntax

```
D = directivity(H,FREQ,ANGLE)
D = directivity(H,FREQ,ANGLE,Name,Value)
```

Description

`D = directivity(H,FREQ,ANGLE)` computes the "Directivity" on page 1-1913 of a uniform rectangular array (URA) of antenna or microphone elements, `H`, at frequencies specified by the `FREQ` and in angles of direction specified by the `ANGLE`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`D = directivity(H,FREQ,ANGLE,Name,Value)` computes the directivity with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

H — Uniform rectangular array

System object

Uniform rectangular array specified as a `phased.URA` System object.

Example: `H = phased.URA`

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

ANGLE — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

PropagationSpeed — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

D — Directivity
M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

Examples

Directivity of Uniform Rectangular Array

Compute the directivity of two uniform rectangular arrays (URA). The first array consists of isotropic antenna elements. The second array consists of cosine antenna elements. In addition, compute the directivity of the first array steered to a specific direction.

Array of isotropic antenna elements

First, create a 10-by-10-element URA of isotropic antenna elements spaced one-quarter wavelength apart. Set the signal frequency to 800 MHz.

```
c = physconst('LightSpeed');
fc = 3e8;
lambda = c/fc;
myAntIso = phased.IsotropicAntennaElement;
myArray1 = phased.URA;
myArray1.Element = myAntIso;
myArray1.Size = [10,10];
myArray1.ElementSpacing = [lambda*0.25,lambda*0.25];
ang = [0;0];
d = directivity(myArray1,fc,ang,'PropagationSpeed',c)
```

```
d = 15.7753
```

Array of cosine antenna elements

Next, create a 10-by-10-element URA of cosine antenna elements also spaced one-quarter wavelength apart.

```
myAntCos = phased.CosineAntennaElement('CosinePower',[1.8,1.8]);
myArray2 = phased.URA;
```

```
myArray2.Element = myAntCos;
myArray2.Size = [10,10];
myArray2.ElementSpacing = [lambda*0.25,lambda*0.25];
ang = [0;0];
d = directivity(myArray2,fc,ang,'PropagationSpeed',c)
```

```
d = 19.7295
```

The directivity is increased due to the directivity of the cosine antenna elements.

Steered array of isotropic antenna elements

Finally, steer the isotropic antenna array to 30 degrees in azimuth and examine the directivity at the steered angle.

```
ang = [30;0];
w = steervec(getElementPosition(myArray1)/lambda,ang);
d = directivity(myArray1,fc,ang,'PropagationSpeed',c,...
    'Weights',w)
```

```
d = 15.3309
```

The directivity is maximum in the steered direction and equals the directivity of the unsteered array at boresight.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

See Also

`pattern` | `patternAzimuth` | `patternElevation`

collectPlaneWave

System object: `phased.URA`
Package: `phased`

Simulate received plane waves

Syntax

```
Y = collectPlaneWave(H,X,ANG)
Y = collectPlaneWave(H,X,ANG,FREQ)
Y = collectPlaneWave(H,X,ANG,FREQ,C)
```

Description

`Y = collectPlaneWave(H,X,ANG)` returns the received signals at the sensor array, `H`, when the input signals indicated by `X` arrive at the array from the directions specified in `ANG`.

`Y = collectPlaneWave(H,X,ANG,FREQ)`, in addition, specifies the incoming signal carrier frequency in `FREQ`.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)`, in addition, specifies the signal propagation speed in `C`.

Input Arguments

`H`

Array object.

`X`

Incoming signals, specified as an M-column matrix. Each column of `X` represents an individual incoming signal.

`ANG`

Directions from which incoming signals arrive, in degrees. `ANG` can be either a 2-by-M matrix or a row vector of length M.

If `ANG` is a 2-by-M matrix, each column specifies the direction of arrival of the corresponding signal in `X`. Each column of `ANG` is in the form `[azimuth; elevation]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive.

If `ANG` is a row vector of length M, each entry in `ANG` specifies the azimuth angle. In this case, the corresponding elevation angle is assumed to be 0°.

`FREQ`

Carrier frequency of signal in hertz. `FREQ` must be a scalar.

Default: `3e8`

C

Propagation speed of signal in meters per second.

Default: Speed of light

Output Arguments

Y

Received signals. Y is an N-column matrix, where N is the number of elements in the array H. Each column of Y is the received signal at the corresponding array element, with all incoming signals combined.

Examples

Simulate Received Signal at URA

Simulate two received random signals at a 6-element URA. The array has a rectangular lattice with two elements in the row direction and three elements in the column direction. The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
array = phased.URA([2 3]);
fc = 100e6;
y = collectPlaneWave(array,randn(4,2),[10 30],fc,physconst('LightSpeed'));
```

Algorithms

`collectPlaneWave` modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. This method does not account for the response of individual elements in the array.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `phitheta2azel`

getElementNormal

System object: `phased.URA`
Package: `phased`

Normal vector to array elements

Syntax

```
normvec = getElementNormal(sURA)
normvec = getElementNormal(sURA,elemidx)
```

Description

`normvec = getElementNormal(sURA)` returns the normal vectors of the array elements of the `phased.URA` System object, `sURA`. The output argument `normvec` is a 2-by-*N* matrix, where *N* is the number of elements in array, `sURA`. Each column of `normvec` defines the normal direction of an element in the local coordinate system in the form`[az;el]`. Units are degrees. Array elements are located in the plane selected in the `ArrayNormal` property. Element normal vectors are parallel to the array normal. The normal to a URA array depends upon the selected `ArrayNormal` property.

ArrayNormal Property Value	Array Normal Direction	Array Plane
`'x'`	azimuth = 0°, elevation = 0° (*x*-axis)	*yz*
`'y'`	azimuth = 90°, elevation = 0° (*y*-axis)	*zx*
`'z'`	azimuth = 0°, elevation = 90° (*z*-axis)	*xy*

The origin of the local coordinate system is defined by the phase center of the array.

`normvec = getElementNormal(sURA,elemidx)` returns only the normal vectors of the elements specified in the element index vector, `elemidx`. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

`sURA` — Uniform rectangular array
`phased.sURA` System object

Uniform line array, specified as a `phased.URA` System object.

Example: `sULA = phased.URA`

`elemidx` — Element indices
all array elements (default) | integer-valued 1-by-*M* row vector | integer-valued *M*-by-1 column vector

Element indices , specified as a 1-by-*M* or *M*-by-1 vector. Index values lie in the range 1 to *N* where *N* is the number of elements of the array. When `elemidx` is specified, `getElementNormal` returns the normal vectors of the elements contained in `elemidx`.

Example: [1,5,4]

Output Arguments

normvec — Element normal vectors
2-by-*P* real-valued vector

Element normal vectors, specified as a 2-by-*P* real-valued vector. Each column of `normvec` takes the form `[az,el]`. When `elemidx` is not specified, *P* equals the array dimension. When `elemidx` is specified, *P* equals the length of `elemidx`, *M*. You can determine element indices using the `viewArray` method.

Examples

URA Element Normals

Construct three 2-by-2 URA's with element normals along the *x*-, *y*-, and *z*-axes. Obtain the element positions and normal directions.

First, choose the array normal along the *x*-axis.

```
sURA1 = phased.URA('Size',[2,2],'ArrayNormal','x');
pos = getElementPosition(sURA1)
```

```
pos = 3×4

         0         0         0         0
   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
```

```
normvec = getElementNormal(sURA1)
```

```
normvec = 2×4

     0     0     0     0
     0     0     0     0
```

All elements lie in the *yz*-plane and the element normal vectors point along the *x*-axis *(0°,0°)*.

Next, choose the array normal along the *y*-axis.

```
sURA2 = phased.URA('Size',[2,2],'ArrayNormal','y');
pos = getElementPosition(sURA2)
```

```
pos = 3×4

    0.2500    0.2500   -0.2500   -0.2500
         0         0         0         0
    0.2500   -0.2500    0.2500   -0.2500
```

```
normvec = getElementNormal(sURA2)
```

```
normvec = 2×4

    90    90    90    90
     0     0     0     0
```

All elements lie in the *zx*-plane and the element normal vectors point along the *y*-axis *(90°,0°)*.

Finally, set the array normal along the *z*-axis. Obtain the normal vectors of the odd-numbered elements.

```
sURA3 = phased.URA('Size',[2,2],'ArrayNormal','z');
pos = getElementPosition(sURA3)
```

```
pos = 3×4

   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
         0         0         0         0
```

```
normvec = getElementNormal(sURA3,[1,3])
```

```
normvec = 2×2

     0     0
    90    90
```

All elements lie in the *xy*-plane and the element normal vectors point along the *z*-axis *(0°,90°)*.

Version History

Introduced in R2016a

getElementPosition

System object: phased.URA
Package: phased

Positions of array elements

Syntax

```
POS = getElementPosition(H)
POS = getElementPosition(H,ELEIDX)
```

Description

POS = getElementPosition(H) returns the element positions of the URA H. POS is a 3-by-N matrix where N is the number of elements in H. Each column of POS defines the position of an element in the local coordinate system, in meters, using the form [x; y; z].

For details regarding the local coordinate system of the URA, enter phased.URA.coordinateSystemInfo.

POS = getElementPosition(H,ELEIDX) returns the positions of the elements that are specified in the element index vector, ELEIDX. The index of a URA runs down each column, then to the next column to the right. For example, in a URA with 4 elements in each row and 3 elements in each column, the element in the third row and second column has an index value of 6.

Examples

Obtain URA Element Positions

Construct a default URA with a rectangular lattice, and obtain the element positions.

```
array = phased.URA;
pos = getElementPosition(array)
```

```
pos = 3×4

         0         0         0         0
   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
```

getNumElements

System object: `phased.URA`
Package: `phased`

Number of elements in array

Syntax

`N = getNumElements(H)`

Description

`N = getNumElements(H)` returns the number of elements, N, in the URA object `H`.

Examples

Obtain Number of URA Elements

Construct a default URA, and obtain the number of elements.

```
array = phased.URA;
N = getNumElements(array)
```

```
N = 4
```

getTaper

System object: phased.URA
Package: phased

Array element tapers

Syntax

```
wts = getTaper(h)
```

Description

wts = getTaper(h) returns the tapers, wts, applied to each element of the phased uniform rectangular array (URA), h. Tapers are often referred to as weights.

Input Arguments

h — Uniform rectangular array
phased.URA System object

Uniform rectangular array specified as aphased.URA System object.

Output Arguments

wts — Array element tapers
N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1, complex-valued vector, where *N* is the number of elements in the array.

Examples

Create Tapered URA

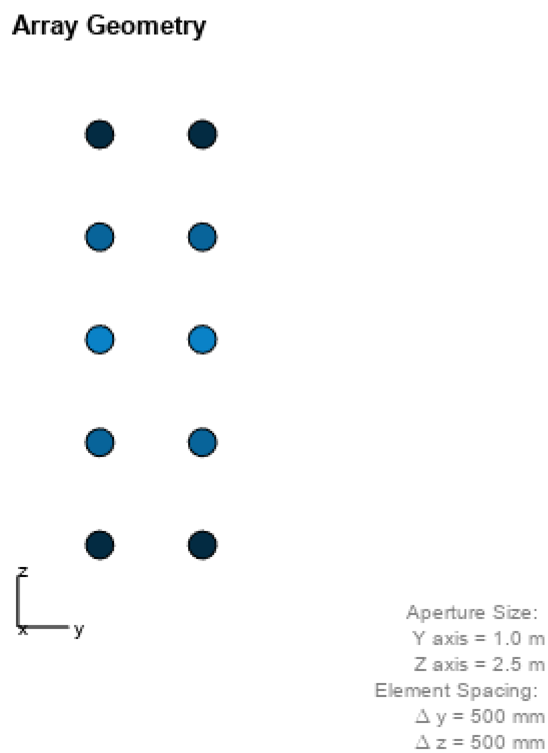
Construct a 5-by-2 element URA with a Taylor window taper along each column. Then, draw the array showing the element taper shading.

```
taper = taylorwin(5);
array = phased.URA([5,2],'Taper',[taper,taper]);
w = getTaper(array)
```

```
w = 10×1

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
    0.5181
```

```
    1.2029
    1.5581
    1.2029
    0.5181
```

```
viewArray(array,'ShowTaper',true)
```



pattern

System object: phased.URA
Package: phased

Plot URA array pattern

Syntax

```
pattern(sArray,FREQ)
pattern(sArray,FREQ,AZ)
pattern(sArray,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

pattern(sArray,FREQ) plots the 3-D array directivity pattern (in dBi) for the array specified in sArray. The operating frequency is specified in FREQ.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

pattern(sArray,FREQ,AZ) plots the array directivity pattern at the specified azimuth angle.

pattern(sArray,FREQ,AZ,EL) plots the array directivity pattern at specified azimuth and elevation angles.

pattern(___ ,Name,Value) plots the array pattern with additional options specified by one or more Name,Value pair arguments.

[PAT,AZ_ANG,EL_ANG] = pattern(___) returns the array pattern in PAT. The AZ_ANG output contains the coordinate values corresponding to the rows of PAT. The EL_ANG output contains the coordinate values corresponding to the columns of PAT. If the 'CoordinateSystem' parameter is set to 'uv', then AZ_ANG contains the *U* coordinates of the pattern and EL_ANG contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Note This method replaces the plotResponse method. See "Convert plotResponse to pattern" on page 1-1930 for guidelines on how to use pattern in place of plotResponse.

Input Arguments

sArray — Uniform rectangular array

System object

Uniform rectangular array, specified as a phased.URA System object.

Example: sArray= phased.URA;

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of 'CoordinateSystem' and one of 'polar', 'rectangular', or 'uv'. When 'CoordinateSystem' is set to 'polar' or 'rectangular', the AZ and EL arguments specify the pattern azimuth and elevation, respectively. AZ values must lie between –180° and 180°. EL values must lie between –90° and 90°. If 'CoordinateSystem' is set to 'uv', AZ and EL then specify *U* and *V* coordinates, respectively. AZ and EL must lie between -1 and 1.

Example: 'uv'

Data Types: char

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Orientation — Array orientation
[0;0;0]. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

ShowArray — View array geometry
false (default) | true

View the array geometry along with the 3D radiation pattern, specified as false or true.

Data Types: logical

ShowLocalCoordinates — Show local coordinate axes
true (default) | false

Show the local coordinate axes, specified as true or false.

Data Types: logical

ShowColorbar — Show colorbar
true (default) | false

Show the colorbar, specified as `true` or `false`.

Data Types: `logical`

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

PlotStyle — Plotting style
`'overlay'` (default) | `'waterfall'`

Plotting style, specified as the comma-separated pair consisting of `'Plotstyle'` and either `'overlay'` or `'waterfall'`. This parameter applies when you specify multiple frequencies in `FREQ` in 2-D plots. You can draw 2-D plots by setting one of the arguments `AZ` or `EL` to a scalar.

Data Types: `char`

Polarization — Polarized field component
`'combined'` (default) | `'H'` | `'V'`

Polarized field component to display, specified as the comma-separated pair consisting of 'Polarization' and `'combined'`, `'H'`, or `'V'`. This parameter applies only when the sensors are polarization-capable and when the `'Type'` parameter is not set to `'directivity'`. This table shows the meaning of the display options.

'Polarization'	Display
`'combined'`	Combined *H* and *V* polarization components
`'H'`	*H* polarization component
`'V'`	*V* polarization component

Example: `'V'`

Data Types: `char`

PropagationSpeed — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`PAT` — Array pattern
M-by-*N* real-valued matrix

Array pattern, returned as an *M*-by-*N* real-valued matrix. The dimensions of `PAT` correspond to the dimensions of the output arguments `AZ_ANG` and `EL_ANG`.

`AZ_ANG` — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in `AZ`. The columns of `PAT` correspond to the values in `AZ_ANG`. Units are in degrees.

`EL_ANG` — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

Examples

Pattern of 5x7-Element URA Antenna Array

Create a 5x7-element URA operating at 1 GHz. Assume the elements are spaced one-half wavelength apart. Show the 3-D array patterns.

Create the array

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[50e6,1000e6],...
    'AxisDirection','Z');
fc = 500e6;
c = physconst('LightSpeed');
lam = c/fc;
sURA = phased.URA('Element',sSD,...
    'Size',[5,7],...
    'ElementSpacing',0.5*lam);
```

Call the `step` method

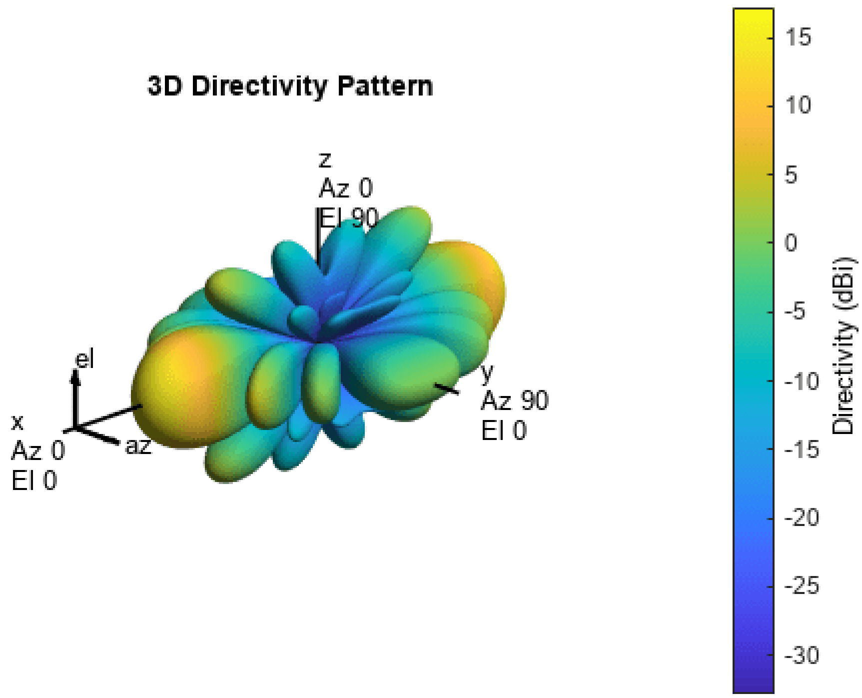
Evaluate the fields of the first five elements at 45 degrees azimuth and 0 degrees elevation.

```
ang = [45;0];
resp = step(sURA,fc,ang);
disp(resp.V(1:5))
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

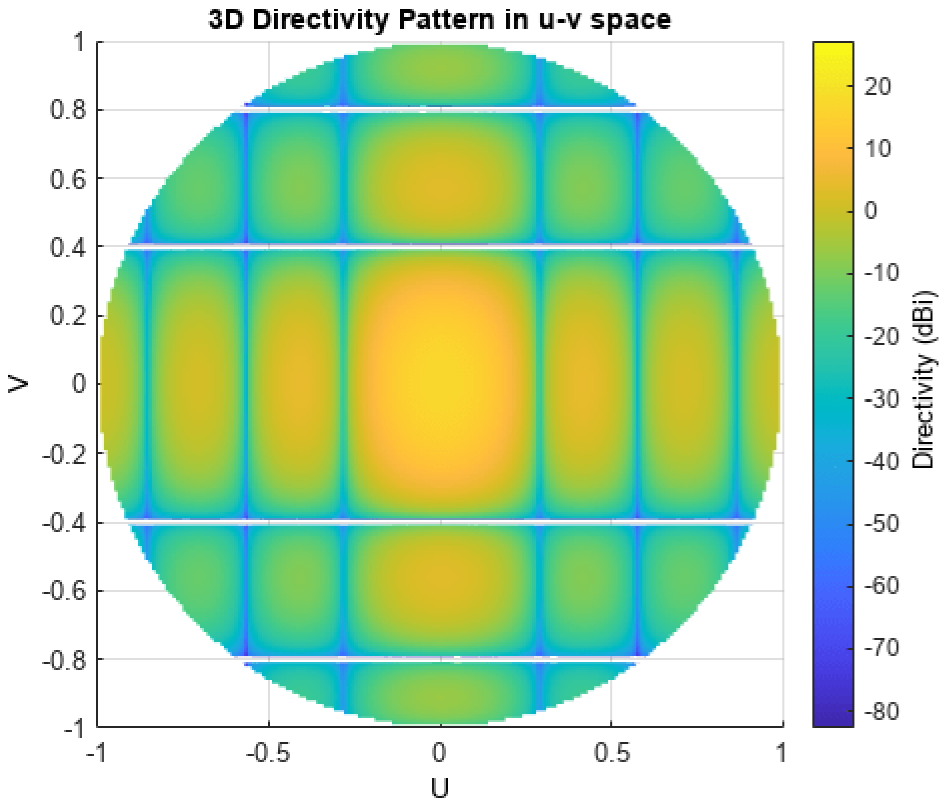
Display the 3-D directivity pattern at 1 GHz in polar coordinates

```
pattern(sURA,fc,[-180:180],[-90:90],...
    'CoordinateSystem','polar',...
    'Type','directivity','PropagationSpeed',c)
```



Display the 3-D directivity pattern at 1 GHz in UV coordinates

```
pattern(sURA,fc,[-1.0:.01:1.0],[-1.0:.01:1.0],...
    'CoordinateSystem','uv',...
    'Type','directivity','PropagationSpeed',c)
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{rad}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Convert plotResponse to pattern

For antenna, microphone, and array System objects, the `pattern` method replaces the `plotResponse` method. In addition, two new simplified methods exist just to draw 2-D azimuth and elevation pattern plots. These methods are the `azimuthPattern` and `elevationPattern` methods.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.

Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.

Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.

'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the UV grid values using AZ and EL.
</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
`'ElevationAngles'` name-value pair	Elevation angles used to display the antenna or array response.	`EL` argument
`'UGrid'` name-value pair	Contains *U* coordinates in *UV*-space.	`AZ` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`
`'VGrid'` name-value pair	Contains *V*-coordinates in *UV*-space.	`EL` argument when `'CoordinateSystem'` name-value pair is set to `'uv'`

Version History

Introduced in R2015a

See Also

`patternAzimuth` | `patternElevation`

patternAzimuth

System object: `phased.URA`
Package: `phased`

Plot URA array directivity or pattern versus azimuth

Syntax

```
patternAzimuth(sArray,FREQ)
patternAzimuth(sArray,FREQ,EL)
patternAzimuth(sArray,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(sArray,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(sArray,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the array `sArray` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(sArray,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`sArray` — Uniform rectangular array

System object

Uniform rectangular array, specified as a `phased.URA` System object.

Example: `sArray= phased.URA;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of 'Weights' and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the phased.SteeringVector System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as phased.Radiator or phased.Collector. However, for the directivity, pattern, patternAzimuth, and patternElevation methods of any array System object use the steering vector without conjugation.

Example: 'Weights',ones(10,1)

Data Types: double
Complex Number Support: Yes

Azimuth — Azimuth angles
[-180:180] (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of 'Azimuth' and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: 'Azimuth',[-90:2:90]

Data Types: double

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the 'Azimuth' name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the EL input argument.

Examples

Azimuth Pattern of 5x7-Element URA Antenna Array

Create a 5x7-element URA of short-dipole antenna elements operating at 1 GHz. Assume the elements are spaced one-half wavelength apart. Plot the array azimuth directivity patterns for two different elevation angles, 0 and 15 degrees. The patternAzimuth method always plots the array pattern in polar coordinates.

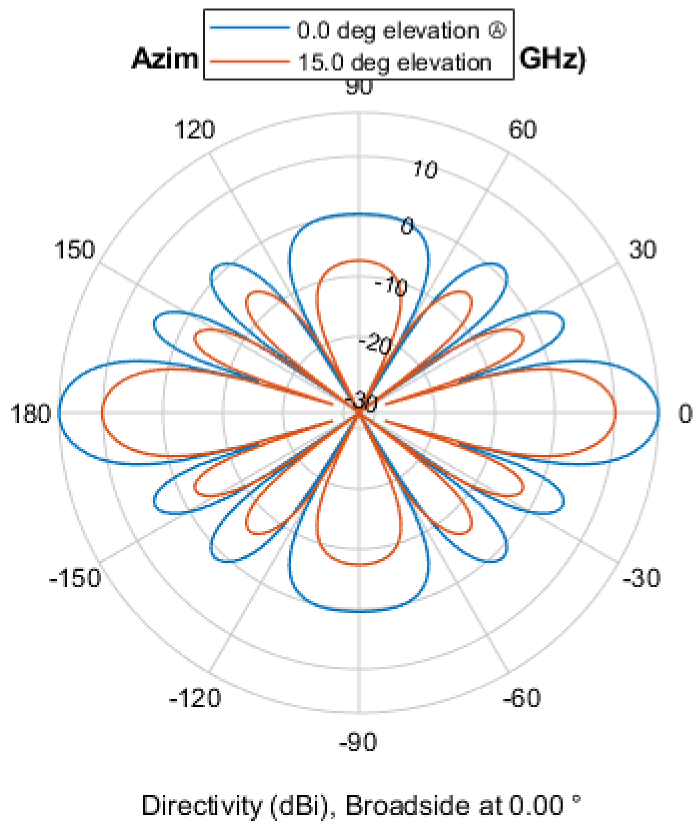
Create the array

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[50e6,1000e6],...
    'AxisDirection','Z');
fc = 1e9;
c = physconst('LightSpeed');
lam = c/fc;
sURA = phased.URA('Element',sSD,...
    'Size',[5,7],...
    'ElementSpacing',0.5*lam);
```

Display the pattern

Display the azimuth directivity pattern at 1 GHz in polar coordinates

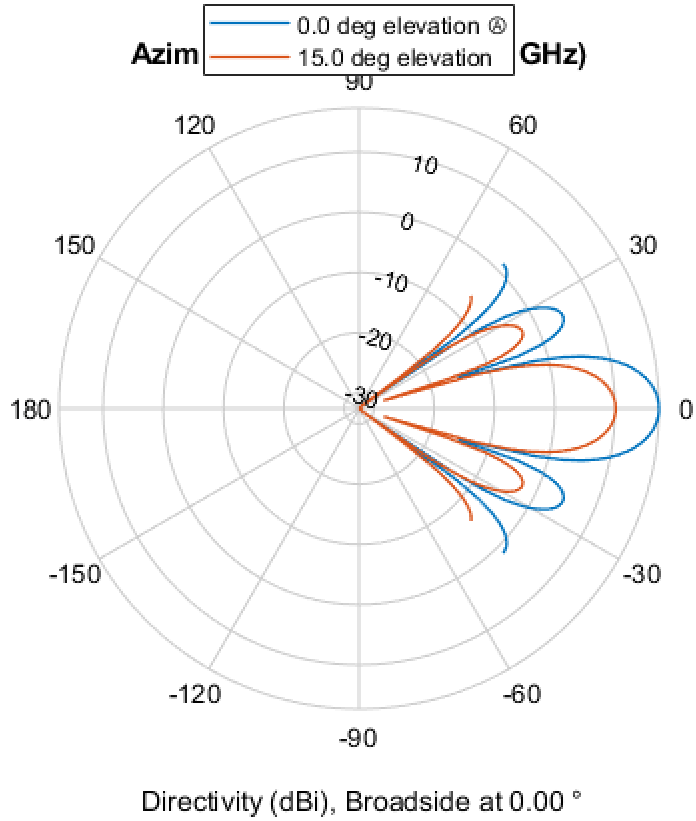
```
patternAzimuth(sURA,fc,[0 15],...
    'PropagationSpeed',c,...
    'Type','directivity')
```



Display a subset of angles

You can plot a smaller range of azimuth angles by setting the `Azimuth` parameter.

```
patternAzimuth(sURA,fc,[0 15],...
    'PropagationSpeed',c,...
    'Type','directivity',...
    'Azimuth',[-45:45])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also
`pattern` | `patternElevation`

patternElevation

System object: `phased.URA`
Package: `phased`

Plot URA array directivity or pattern versus elevation

Syntax

```
patternElevation(sArray,FREQ)
patternElevation(sArray,FREQ,AZ)
patternElevation(sArray,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(sArray,FREQ)` plots the 2-D array directivity pattern versus elevation (in dBi) for the array `sArray` at zero degrees azimuth angle. When `AZ` is a vector, multiple overlaid plots are created. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternElevation(sArray,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(sArray,FREQ,AZ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`sArray` — Uniform rectangular array

System object

Uniform rectangular array, specified as a `phased.URA` System object.

Example: `sArray= phased.URA;`

`FREQ` — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element

produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `1e8`

Data Types: `double`

`AZ` — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of 'Weights' and an *M*-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *M* is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the phased.SteeringVector System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as phased.Radiator or phased.Collector. However, for the directivity, pattern, patternAzimuth, and patternElevation methods of any array System object use the steering vector without conjugation.

Example: 'Weights',ones(10,1)

Data Types: double
Complex Number Support: Yes

Elevation — Elevation angles
[-90:90] (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of 'Elevation' and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: 'Elevation',[-90:2:90]

Data Types: double

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of elevation angles determined by the 'Elevation' name-value pair argument. The dimension *N* is the number of azimuth angles determined by the AZ argument.

Examples

Elevation Pattern of 7x7-Element Acoustic URA

Create a 7x7-element URA of backbaffled omnidirectional transducer elements operating at 2 kHz. Assume the speed of sound in water is 1500 m/s. The elements are spaced less than one-half wavelength apart. Plot the array elevation directivity patterns for three different azimuth angles, -20, 0, and 15 degrees. The patternElevation method always plots the array pattern in polar coordinates.

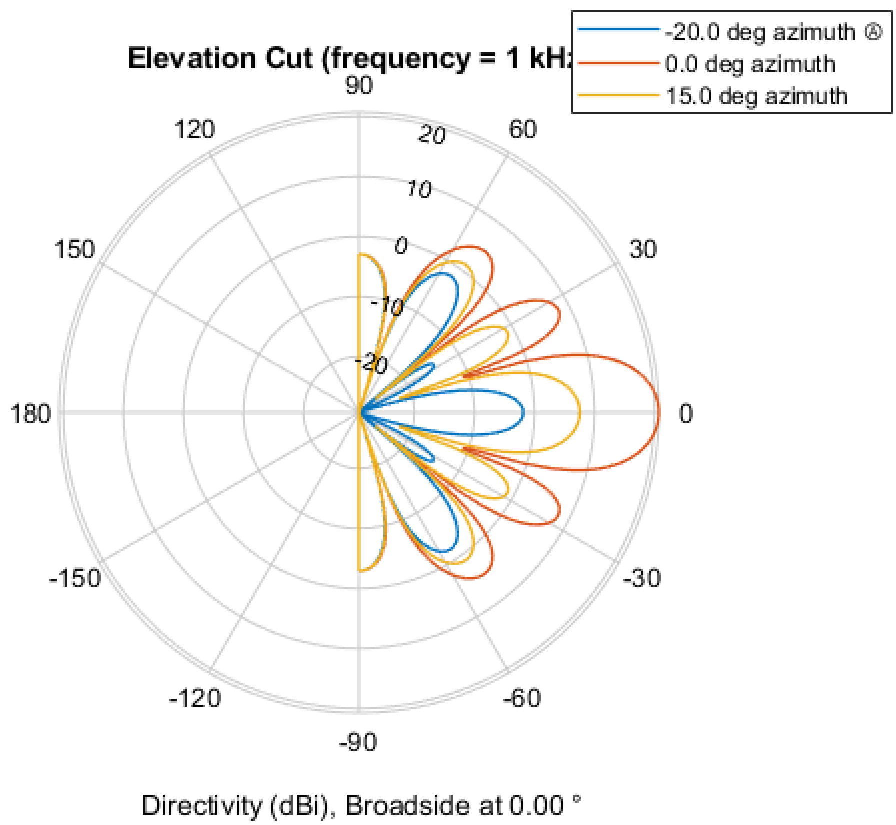
Create the array

```
element = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20,3000],...
    'BackBaffled',true);
fc = 1000;
c = 1500;
lam = c/fc;
array = phased.URA('Element',element,...
    'Size',[7,7],...
    'ElementSpacing',0.45*lam);
```

Display the pattern

Display the azimuth directivity pattern at 1 GHz in polar coordinates.

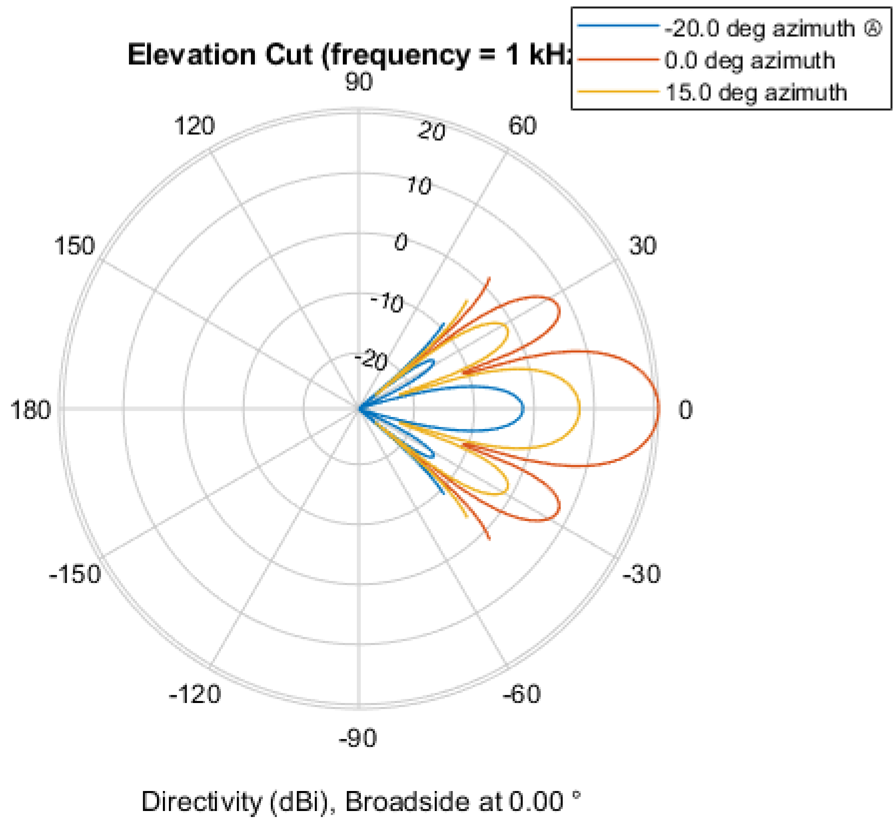
```
patternElevation(array,fc,[-20, 0, 15],...
    'PropagationSpeed',c,...
    'Type','directivity')
```



Display a subset of elevation angles

You can plot a smaller range of elevation angles by setting the `Elevation` parameter.

```
patternElevation(array,fc,[-20, 0, 15],...
    'PropagationSpeed',c,...
    'Type','directivity',...
    'Elevation',[-45:45])
```



More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\text{rad}}(\theta, \varphi)}{P_{\text{total}}}$$

where $U_{\text{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Version History

Introduced in R2015a

See Also
pattern | patternAzimuth

isPolarizationCapable

System object: phased.URA
Package: phased

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h — Uniform rectangular array

Uniform rectangular array specified as `phased.URA` System object.

Output Arguments

flag — Polarization-capability flag

Polarization-capability flag returned as a Boolean value, `true`, if the array supports polarization or, `false`, if it does not.

Examples

Short-Dipole URA Supports Polarization

Show that a URA array of `phased.ShortDipoleAntennaElement` short-dipole antenna elements supports polarization.

```
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[1e9 10e9]);
array = phased.URA([3,2],'Element',antenna);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

The returned value `1` shows that this array supports polarization.

plotResponse

System object: `phased.URA`
Package: `phased`

Plot response pattern of array

Syntax

```
plotResponse(H,FREQ,V)
plotResponse(H,FREQ,V,Name,Value)
hPlot = plotResponse( ___ )
```

Description

`plotResponse(H,FREQ,V)` plots the array response pattern along the azimuth cut, where the elevation angle is 0. The operating frequency is specified in `FREQ`. The propagation speed is specified in `V`.

`plotResponse(H,FREQ,V,Name,Value)` plots the array response with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = plotResponse( ___ )` returns handles of the lines or surface in the figure window, using any of the input arguments in the previous syntaxes.

Input Arguments

`H`

Array object

`FREQ`

Operating frequency in Hertz specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

`V`

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`CutAngle`

Cut angle as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

`Format`

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

`NormalizeResponse`

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

`OverlayFreq`

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

`Polarization`

Specify the polarization options for plotting the array response pattern. The allowable values are `|'None' | 'Combined' | 'H' | 'V' |` where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

`RespCut`

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.

- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

Unit

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

Weights

Weight values applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. The dimension *N* is the number of elements in the array. The interpretation of *M* depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

AzimuthAngles

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

ElevationAngles

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the

`Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

`UGrid`

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

`VGrid`

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

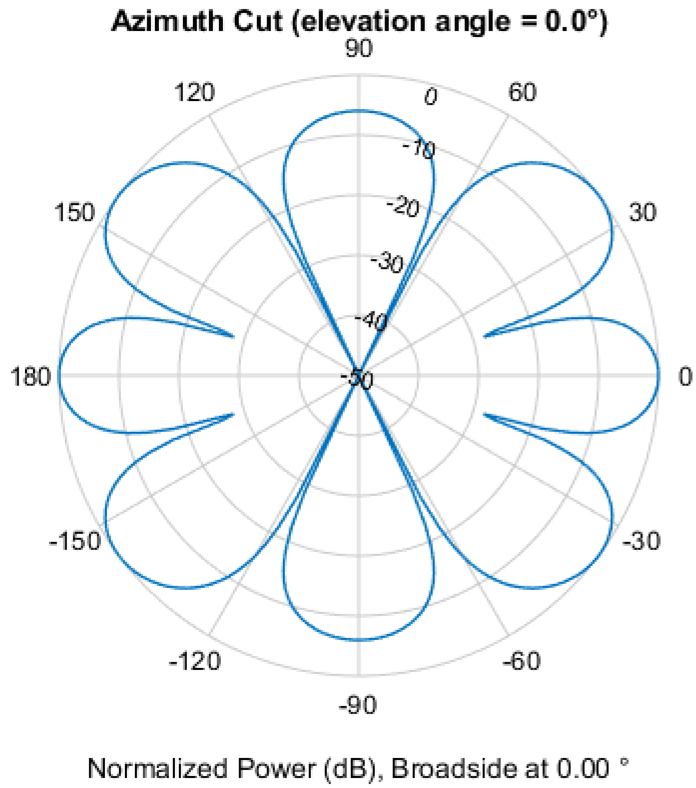
Default: `[-1:0.01:1]`

Examples

Azimuth Response of URA

This example shows how to construct a rectangular lattice 3-by-2 URA and plot that array's azimuth response.

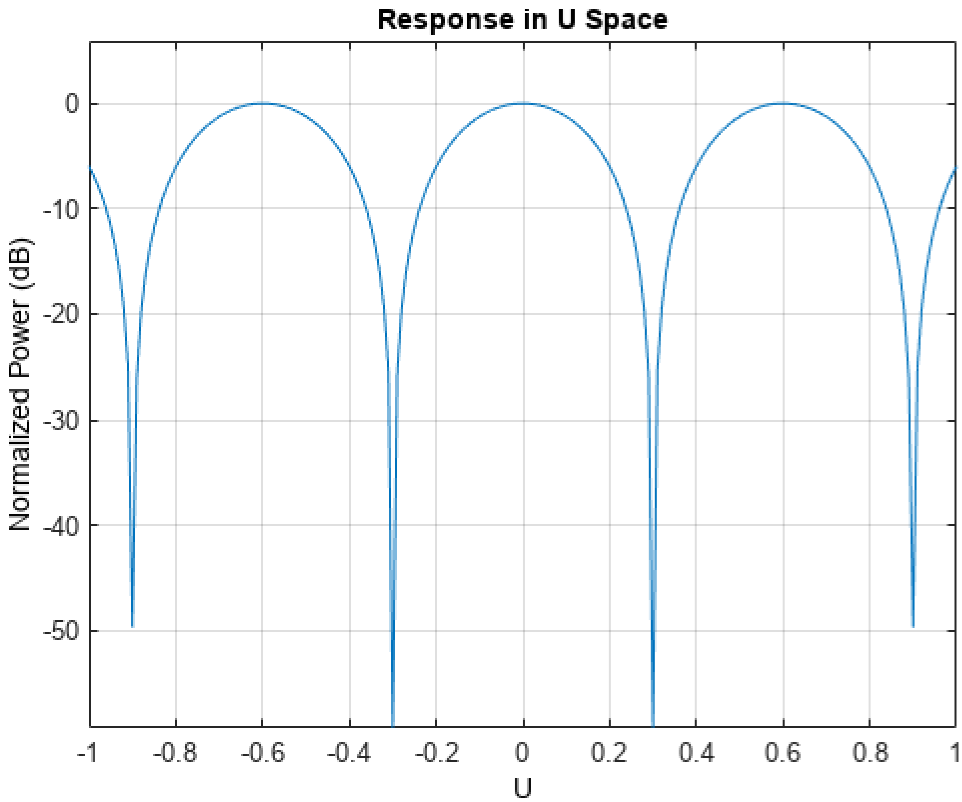
```
ha = phased.URA('Size',[3 2]);
fc = 1e9;
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



Array Response and Directivity of URA in U/V Space

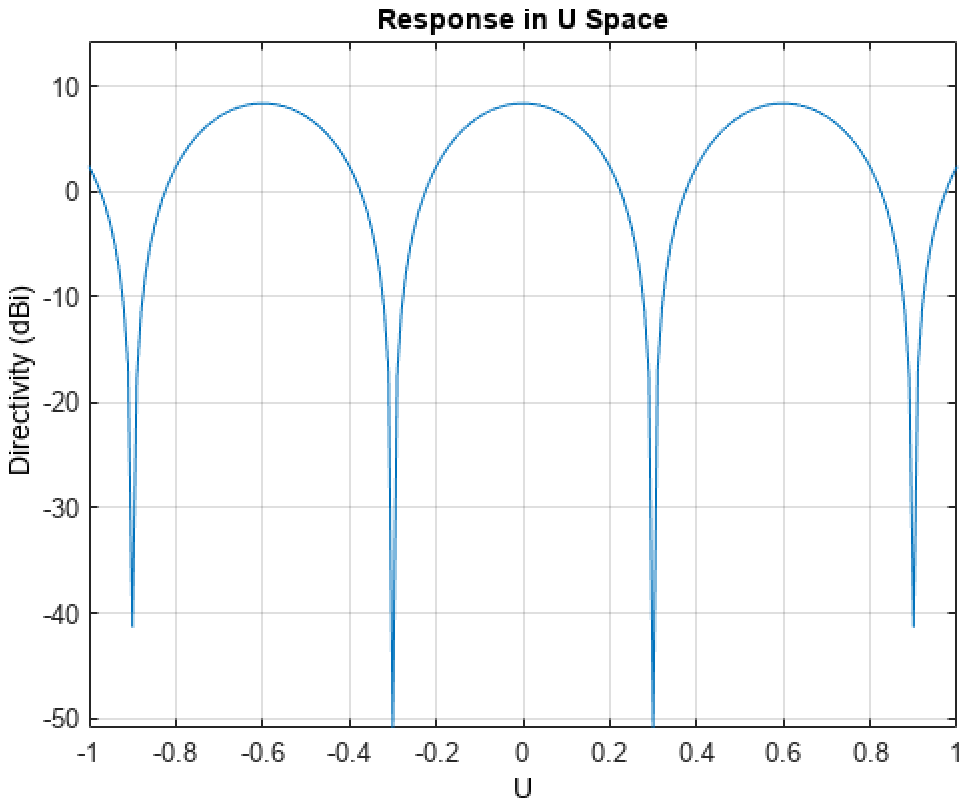
This example shows how to construct a rectangular lattice 3-by-2 URA. Plot the u cut of the array response in $u-v$ space.

```
ha = phased.URA('Size',[3 2]);
c = physconst('lightspeed');
plotResponse(ha,1e9,c,'Format','UV');
```



Plot the directivity.

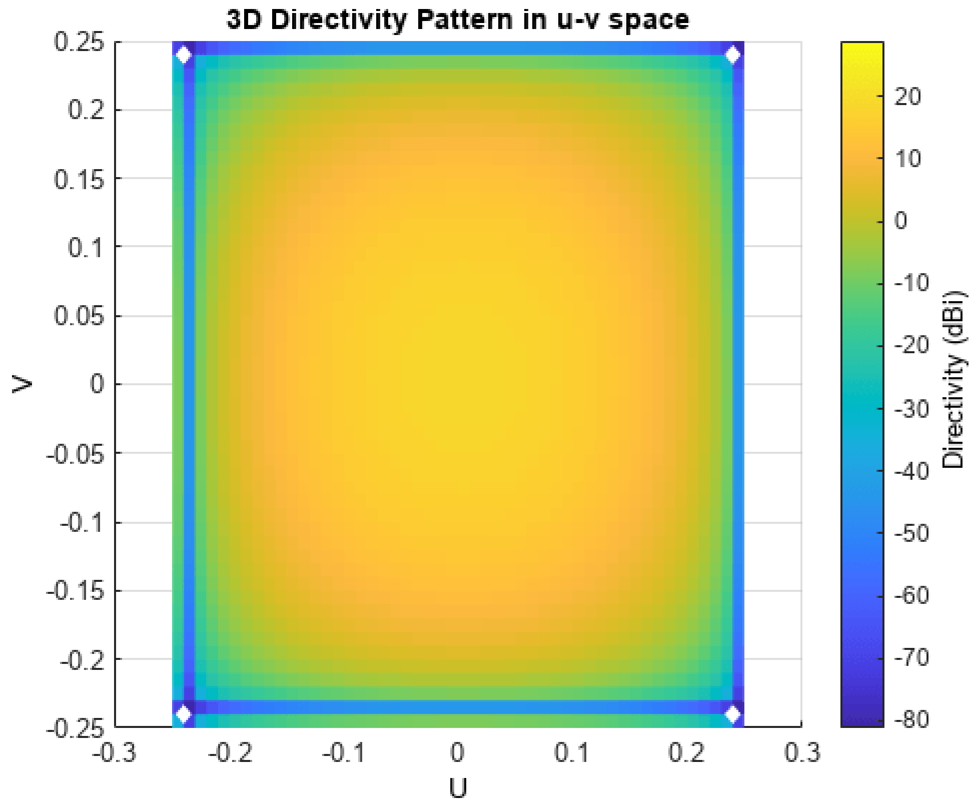
```
plotResponse(ha,1e9,c,'Format','UV','Unit','dbi');
```



Array Response of URA for Subrange of U-V Space

Construct a 5-by-5 square uniform rectangular array (URA) and then plot the 3D response in *u*-*v* space. Restrict the *u*-*v* range from -0.25 to 0.25.

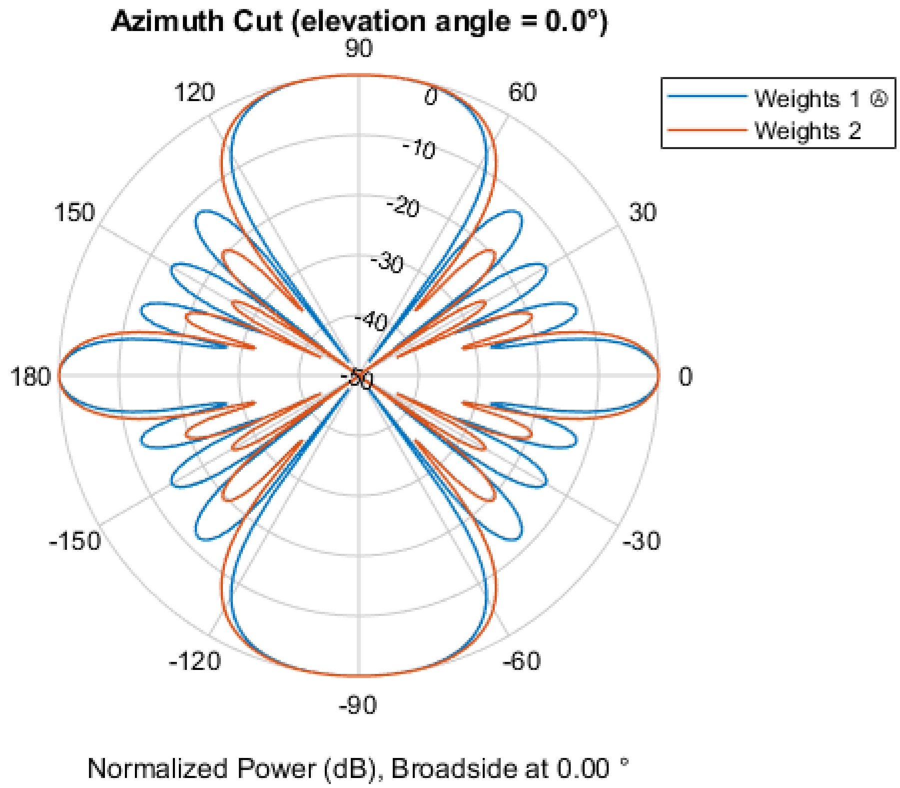
```
array = phased.URA([5,5]);
fc = 500e6;
c = physconst('LightSpeed');
pattern(array,fc,[-0.25:.01:.25],[-0.25:.01:.25],'PropagationSpeed',c,'CoordinateSystem','uv')
```



Array Response of URA with Two Sets of Weights

This example shows how to construct a square 5-by-5 URA array having elements spaced 0.3 meters apart. Apply both uniform weights and tapered weights at a single frequency using the `Weights` parameter. Choose the tapered weight values to be smallest at the edges and increasing towards the center. Then, show that the tapered weight set reduces the adjacent sidelobes while broadening the main lobe.

```
ha = phased.URA('Size',[5 5],'ElementSpacing',[0.3,0.3]);
fc = 1e9;
c = physconst('LightSpeed');
wts1 = ones(5,5);
wts1 = wts1(:);
wts1 = wts1/sum(wts1);
wts2 = 0.3*ones(5,5);
wts2(2:4,2:4) = 0.7;
wts2(3,3) = 1;
wts2 = wts2(:);
wts2 = wts2/sum(wts2);
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar','Weights',[wts1,wts2]);
```



See Also

uv2azel | azel2uv

plotGratingLobeDiagram

System object: `phased.URA`
Package: `phased`

Plot grating lobe diagram of array

Syntax

```
plotGratingLobeDiagram(H,FREQ)
plotGratingLobeDiagram(H,FREQ,ANGLE)
plotGratingLobeDiagram(H,FREQ,ANGLE,C)
plotGratingLobeDiagram(H,FREQ,ANGLE,C,F0)
hPlot = plotGratingLobeDiagram( ___ )
```

Description

`plotGratingLobeDiagram(H,FREQ)` plots the grating lobe diagram of an array in the *u-v* coordinate system. The System object `H` specifies the array. The argument `FREQ` specifies the signal frequency and phase-shifter frequency. The array, by default, is steered to 0° azimuth and 0° elevation.

A grating lobe diagram displays the positions of the peaks of the narrowband array pattern. The array pattern depends only upon the geometry of the array and not upon the types of elements which make up the array. Visible and nonvisible grating lobes are displayed as open circles. Only grating lobe peaks near the location of the mainlobe are shown. The mainlobe itself is displayed as a filled circle.

`plotGratingLobeDiagram(H,FREQ,ANGLE)`, in addition, specifies the array steering angle, `ANGLE`.

`plotGratingLobeDiagram(H,FREQ,ANGLE,C)`, in addition, specifies the propagation speed by `C`.

`plotGratingLobeDiagram(H,FREQ,ANGLE,C,F0)`, in addition, specifies an array phase-shifter frequency, `F0`, that differs from the signal frequency, `FREQ`. This argument is useful when the signal no longer satisfies the narrowband assumption and, allows you to estimate the size of beam squint.

`hPlot = plotGratingLobeDiagram( ___ )` returns the handle to the plot for any of the input syntax forms.

Input Arguments

`H`

Antenna or microphone array, specified as a System object.

`FREQ`

Signal frequency, specified as a scalar. Frequency units are hertz. Values must lie within a range specified by the frequency property of the array elements contained in `H.Element`. The frequency property is named `FrequencyRange` or `FrequencyVector`, depending on the element type.

ANGLE

Array steering angle, specified as either a 2-by-1 vector or a scalar. If `ANGLE` is a vector, it takes the form `[azimuth;elevation]`. The azimuth angle must lie in the range `[-180°,180°]`. The elevation angle must lie in the range `[-90°,90°]`. All angle values are specified in degrees. If the argument `ANGLE` is a scalar, it specifies only the azimuth angle where the corresponding elevation angle is 0°.

Default: `[0;0]`

C

Signal propagation speed, specified as a scalar. Units are meters per second.

Default: Speed of light in vacuum

F0

Phase-shifter frequency of the array, specified as a scalar. Frequency units are hertz When this argument is omitted, the phase-shifter frequency is assumed to be the signal frequency, `FREQ`.

Default: `FREQ`

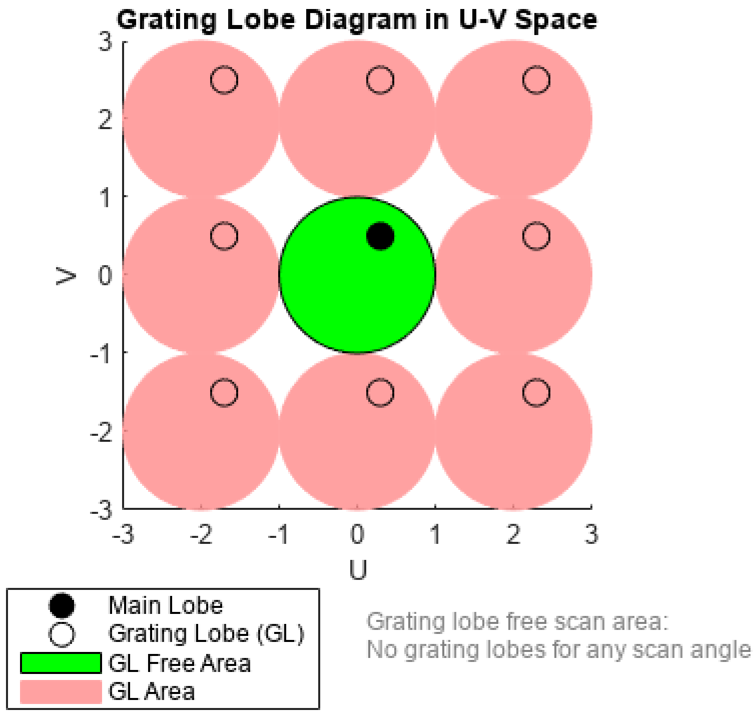
Examples

Grating Lobe Diagram for Microphone URA

Plot the grating lobe diagram for an 11-by-9-element uniform rectangular array having element spacing equal to one-half wavelength.

Assume the operating frequency of the array is 10 kHz. All elements are omnidirectional microphone elements. Steer the array in the direction 20 degrees in azimuth and 30 degrees in elevation. The speed of sound in air is 344.21 m/s at 21 deg C.

```
cair = 344.21;
f = 10.0e3;
lambda = cair/f;
microphone = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
array = phased.URA('Element',microphone,'Size',[11,9],...
    'ElementSpacing',0.5*lambda*[1,1]);
plotGratingLobeDiagram(array,f,[20;30],cair);
```



Plot the grating lobes. The main lobe of the array is indicated by a filled black circle. The grating lobes in visible and nonvisible regions are indicated by unfilled black circles. The visible region is the region in u-v coordinates for which $u^2 + v^2 \leq 1$. The visible region is shown as a unit circle centered at the origin. Because the array spacing is less than one-half wavelength, there are no grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it coincides with the visible region.

The white areas of the diagram indicate a region where no grating lobes are possible.

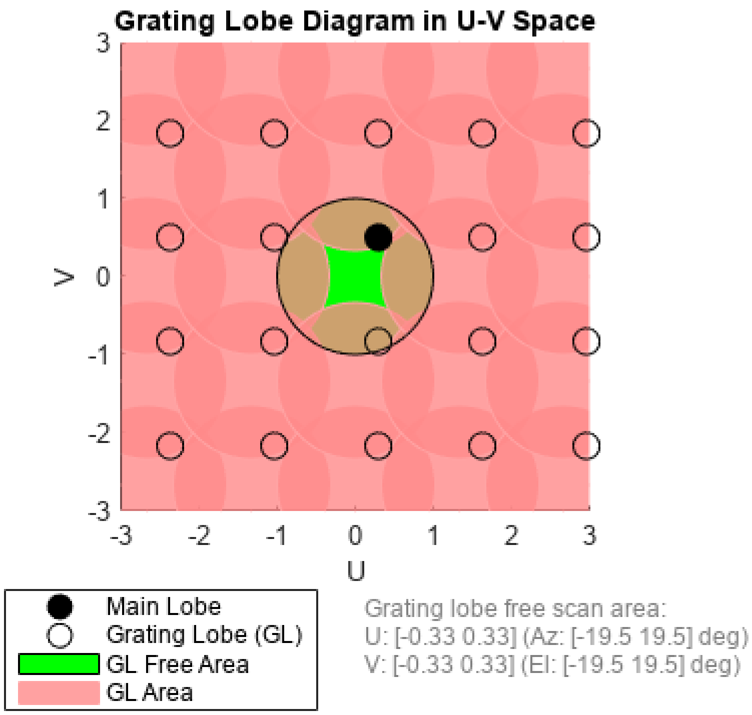
Create Grating Lobe Diagram for Undersampled Microphone URA

Plot the grating lobe diagram for an 11-by-9-element uniform rectangular array having element spacing greater than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 10 kHz and the spacing between elements is 0.75 of a wavelength. All elements are omnidirectional microphone elements. Steer the array in the direction 20 degrees in azimuth and 30 degrees in elevation. The speed of sound in air is 344.21 m/s at 21 deg C.

```
cair = 344.21;
f = 10000;
```

```
lambda = cair/f;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
sURA = phased.URA('Element',sMic,'Size',[11,9],...
    'ElementSpacing',0.75*lambda*[1,1]);
plotGratingLobeDiagram(sURA,f,[20;30],cair);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in visible and nonvisible regions are indicated by unfilled black circles. The visible region is the region in u-v coordinates for which $u^2 + v^2 \leq 1$. The visible region is shown as a unit circle centered at the origin. Because the array spacing is greater than one-half wavelength, there are grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

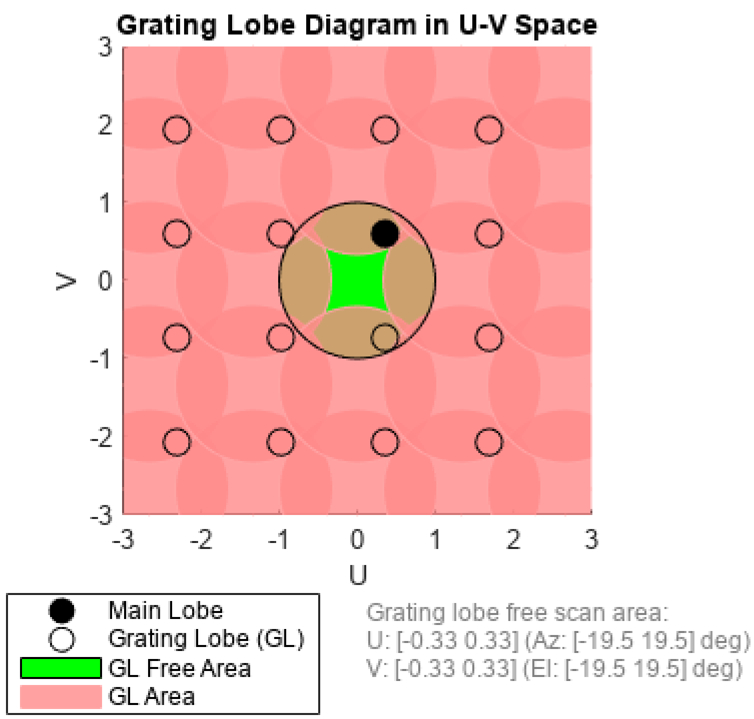
The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region. Because the mainlobe is outside the green area, there is a grating lobe within the visible region.

Create Grating Lobe Diagram for Microphone URA with Frequency Shift

Plot the grating lobe diagram for an 11-by-9-element uniform rectangular array having element spacing greater than one-half wavelength. Apply a 20% phase-shifter frequency offset. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 10 kHz and the spacing between elements is 0.75 of a wavelength. All elements are omnidirectional microphone elements. Steer the array in the direction 20 degrees in azimuth and 30 degrees in elevation. The shifted frequency is 12000 Hz. The speed of sound in air is 344.21 m/s at 21 deg C.

```
cair = 344.21;
f = 10000;
f0 = 12000;
lambda = cair/f;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
sURA = phased.URA('Element',sMic,'Size',[11,9],...
    'ElementSpacing',0.75*lambda*[1,1]);
plotGratingLobeDiagram(sURA,f,[20;30],cair,f0);
```



The mainlobe of the array is indicated by a filled black circle. The mainlobe has moved from its position in the previous example due to the frequency shift. The grating lobes in visible and nonvisible regions are indicated by unfilled black circles. The visible region is the region in u-v coordinates for which $u^2 + v^2 \leq 1$. The visible region is shown as a unit circle centered at the origin. Because the array spacing is greater than one-half wavelength, there are grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region. Because the mainlobe is outside the green area, there is a grating lobe within the visible region.

Concepts

Grating Lobes

Spatial undersampling of a wavefield by an array produces visible grating lobes. If you think of the wavenumber, k, as analogous to angular frequency, then you must sample the signal at spatial intervals smaller than π/k_{max} (or $\lambda_{min}/2$) to remove aliasing. The appearance of visible grating lobes is also known as spatial aliasing. The variable k_{max} is the largest wavenumber value present in the signal.

The directions of maximum spatial response of a URA are determined by the peaks of the array pattern (alternatively called the beam pattern or array factor.) Peaks other than the main lobe peak are called grating lobes. For a URA, the array pattern depends only on the wavenumber component of the wavefield in the array plane (the y and z directions for the `phased.URA` System object). The wavenumber components are related to the look-direction of an arriving wavefield by $k_y = -2\pi \sin az \cos el/\lambda$ and $k_z = -2\pi \sin el/\lambda$. The angle az is azimuth angle of the arriving wavefield. The angle el is elevation angle of the arriving wavefield. The look-direction points away from the array to the wavefield source.

The array pattern possesses an infinite number of periodically spaced peaks that are equal in strength to the mainlobe peak. If you steer the array to the az_0, el_0 azimuth and elevation direction, the array pattern for the URA has its mainlobe peak at the wavenumber value, $k_{y0} = -2\pi \sin az_0 \cos el_0/\lambda$, $k_{z0} = -2\pi \sin el_0/\lambda$. The array pattern has strong peaks at $k_{ym} = k_{y0} + 2\pi\, m/d_y$, $k_{zn} = k_{z0} + 2\pi\, n/d_z$ for integer values of m and n. The quantities d_y and d_z are the inter-element spacings in the y- and z-directions, respectively. Expressed in terms of direction cosines, the grating lobes occur at $u_m = u_0 - m\lambda/d_y$ and $v_n = v_0 - n\lambda/d_z$. The main lobe direction cosines are determined by $u_0 = \sin az_0 \cos el_0$ and $v_0 = \sin el_0$ when expressed in terms of the look-direction.

Grating lobes can be visible or nonvisible, depending upon the value of $u_m^2 + v_n^2$. When $u_m^2 + v_n^2 \leq 1$, the look direction represents a visible direction. When the value is greater than one, the grating lobe is non-visible. For each visible grating lobe, you can compute a look direction $(az_{m,n},el_{m,n})$ from $u_m = \sin az_m \cos el_m$ and $v_n = \sin el_n$. The spacing of grating lobes depends upon λ/d. When λ/d is small enough, multiple grating lobe peaks can correspond to physical look-directions.

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

See Also

`uv2azel` | `azel2uv`

step

System object: `phased.URA`
Package: `phased`

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`RESP = step(H,FREQ,ANG)` returns the responses of the array elements, `RESP`, at the operating frequencies specified in `FREQ` and directions specified in `ANG`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`H`

Array object

`FREQ`

Operating frequencies of array in hertz. `FREQ` is a row vector of length *L*. Typical values are within the range specified by a property of `H.Element`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has zero response at frequencies outside that range.

`ANG`

Directions in degrees. `ANG` is either a 2-by-*M* matrix or a row vector of length *M*.

If `ANG` is a 2-by-*M* matrix, each column of the matrix specifies the direction in the form `[azimuth; elevation]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive.

If ANG is a row vector of length M, each element specifies the azimuth angle of the direction. In this case, the corresponding elevation angle is assumed to be 0°.

Output Arguments

RESP

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, RESP, has the dimensions N-by-M-by-L. N is the number of elements in the array. The dimension M is the number of angles specified in ANG. L is the number of frequencies specified in FREQ. For any element, the columns of RESP contain the responses of the array elements for the corresponding direction specified in ANG. Each of the L pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.
- If the array is capable of supporting polarization, the voltage response, RESP, is a MATLAB struct containing two fields, RESP.H and RESP.V. The field, RESP.H, represents the array's horizontal polarization response, while RESP.V represents the array's vertical polarization response. Each field has the dimensions N-by-M-by-L. N is the number of elements in the array, and M is the number of angles specified in ANG. L is the number of frequencies specified in FREQ. Each column of RESP contains the responses of the array elements for the corresponding direction specified in ANG. Each of the L pages of RESP contains the responses of the array elements for the corresponding frequency specified in FREQ.

Examples

Response of 2-by-2 URA of Short-Dipole Antennas

Construct a 2-by-2 rectangular lattice URA of short-dipole antenna elements. Then, find the response of each element at boresight. Assume the operating frequency is 1 GHz.

```
sSD = phased.ShortDipoleAntennaElement;
sURA = phased.URA('Element',sSD,'Size',[2 2]);
fc = 1e9;
ang = [0;0];
resp = step(sURA,fc,ang);
disp(resp.V)
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

See Also

uv2azel | phitheta2azel

viewArray

System object: `phased.URA`
Package: `phased`

View array geometry

Syntax

```
viewArray(H)
viewArray(H,Name,Value)
hPlot = viewArray( ___ )
```

Description

`viewArray(H)` plots the geometry of the array specified in `H`.

`viewArray(H,Name,Value)` plots the geometry of the array, with additional options specified by one or more `Name,Value` pair arguments.

`hPlot = viewArray( ___ )` returns the handle of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Input Arguments

`H`

Array object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

`Parent`

Handle to the axes along which the array geometry is displayed.

`ShowNormals`

Set this value to `true` to show the normal directions of all elements of the array. Set this value to `false` to plot the elements without showing normal directions.

Default: `false`

`ShowLocalCoordinates`

Logical flag specifying whether to show the local coordinate axes.

Default: `true`

`ShowAnnotation`

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Default: `true`

`Orientation`

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Default: `[0;0;0]`

`ShowTaper`

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

`ShowIndex`

Vector specifying the element indices to show in the figure. Each number in the vector must be an integer between 1 and the number of elements. You can also specify the value `'All'` to show the indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

`Title`

Character vector specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

`hPlot`

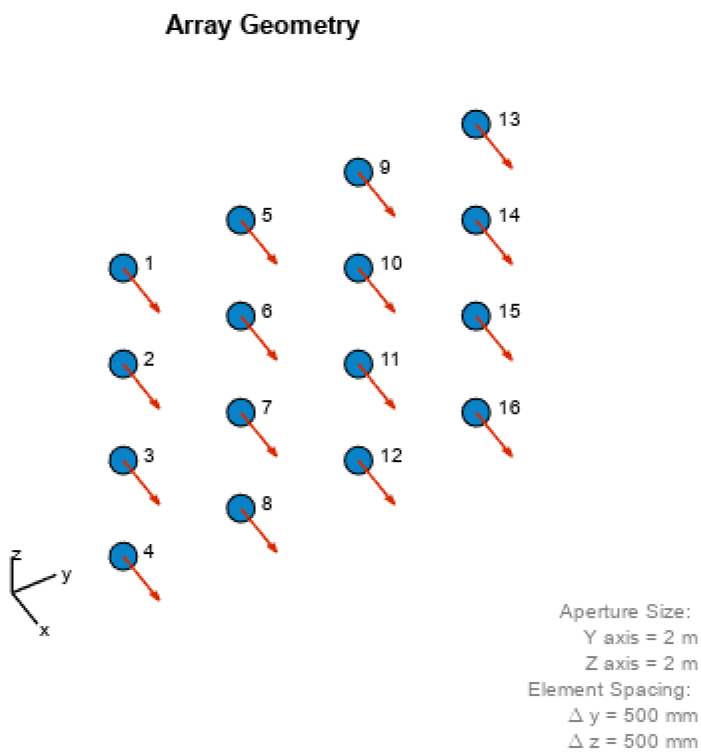
Handle of array elements in figure window.

Examples

Geometry, Normal Directions, and Indices of URA Elements

This example shows how to display the element positions, normal directions, and indices for all elements of a 4-by-4 square URA.

```
ha = phased.URA(4);
viewArray(ha,'ShowNormals',true,'ShowIndex','All');
```



See Also
`phased.ArrayResponse`

Topics
"Phased Array Gallery"

phased.WidebandBackscatterRadarTarget

Package: `phased`

Backscatter wideband signal from radar target

Description

The `phased.WidebandBackscatterRadarTarget` System object models backscattering of a wideband signal from a target. Backscattering is a special case of radar target scattering where the incident and reflected angles are the same. Use this object for monostatic radar configurations. The radar cross-section determines the backscattering response of a target to an incoming signal. This System object lets you specify an angle-dependent radar cross-section model that covers a range of incident angles. The wideband signal is decomposed into frequency subbands which are backscattered independently and then recombined.

This System object creates a backscattered signal for polarized or nonpolarized signals. Although electromagnetic radar signals are polarized, you can often ignore polarization in your simulation and process the signals as scalars. To ignore polarization, specify the `EnablePolarization` property as `false`. To employ polarization, specify `EnablePolarization` as `true`.

For nonpolarized signals, specify the radar cross section (RCS) as an array of values at discrete azimuth and elevation angles and discrete frequencies. The System object interpolates values for incident angles between array points. For polarized signals, specify the radar scattering matrix (SCM) using three arrays defined at discrete azimuth and elevation angles and discrete frequencies. These three arrays correspond to the *HH*, *HV*, and *VV* polarization components. The *VH* component is computed by applying the conjugate symmetry property of the *HV* component. *H* and *V* stand for the horizontal and vertical polarization components, respectively.

For both nonpolarized and polarized signals, you can employ one of four Swerling models to generate random fluctuations in the RCS or radar scattering matrix. Choose the model using the `Model` property. Then, use the `SeedSource` and `Seed` properties to randomize the fluctuations.

EnablePolarization	Radar cross-section patterns
`false`	RCSPattern
`true`	ShhPattern, SvvPattern, and ShvPattern

To perform wideband target backscattering:

1. Create the `phased.WidebandBackscatterRadarTarget` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

`target = phased.WidebandBackscatterRadarTarget`

`target = phased.WidebandBackscatterRadarTarget(Name,Value)`

Description

`target = phased.WidebandBackscatterRadarTarget` creates a wideband backscatter radar `target` System object, .

`target = phased.WidebandBackscatterRadarTarget(Name,Value)` creates a wideband backscatter radar target object, with each specified property `Name` set to the specified `Value`. You can specify additional name and value pair arguments in any order as (`Name1,Value1`,...,`NameN,ValueN`).

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`EnablePolarization` — Enable polarized signals
`false` (default) | `true`

Option to enable processing of polarized signals, specified as `false` or `true`. Set this property to `true` to allow the target to simulate the reflection of polarized radiation. Set this property to `false` to ignore polarization.

Example: `true`

`FrequencyVector` — Wideband backscatter pattern frequencies
`[0,1e20]` (default) | real-valued row vector in strictly increasing order

Specify the wideband backscatter pattern frequencies used in the RCS or SCM matrices. The elements of this vector must be in strictly increasing order. The target has no response outside this frequency range. Frequencies are defined with respect to the physical frequency band, not the baseband. Frequency units are in hertz.

Example: `[1e3,1e10]`

`AzimuthAngles` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector | *P*-by-1 real-valued column vector

Azimuth angles used to define the angular coordinates of each column of the matrices specified by the `RCSPattern`, `ShhPattern`, `ShvPattern`, or `SvvPattern` properties. Specify the azimuth angles as a length *P* vector. *P* must be greater than two. Angle units are in degrees.

Example: `[-45:0.1:45]`

Data Types: `double`

`ElevationAngles` — Elevation angles
`[-90:90]` (default) | 1-by-*Q* real-valued row vector | *Q*-by-1 real-valued column vector

Elevation angles used to define the angular coordinates of each row of the matrices specified by the `RCSPattern`, `ShhPattern`, `ShvPattern`, or `SvvPattern` properties. Specify the elevation angles as a length *Q* vector. *Q* must be greater than two. Angle units are in degrees.

Example: `[-30:0.1:30]`

Data Types: `double`

`RCSPattern` — Radar cross-section pattern

`ones(181,361)` (default) | *Q*-by-*P* real-valued matrix | *Q*-by-*P*-by-*K* real-valued array | 1-by-*P*-by-*K* real-valued array | *K*-by-*P* real-valued matrix

Radar cross-section pattern, specified as a real-valued matrix or array.

<table>
<tr><th>Dimensions</th><th>Application</th></tr>
<tr><td>Q-by-P matrix</td><td>Specifies a matrix of RCS values as a function of Q elevation angles and P azimuth angles. The same RCS matrix is used for all frequencies.</td></tr>
<tr><td>Q-by-P-by-K array</td><td>Specifies an array of RCS patterns as a function of Q elevation angles, P azimuth angles, and K frequencies. If K = 1, the RCS pattern is equivalent to a Q-by-P matrix.</td></tr>
<tr><td>1-by-P-by-K array</td><td rowspan="2">Specifies a matrix of RCS values as a function of P azimuth angles and K frequencies. These dimension formats apply when there is only one elevation angle.</td></tr>
<tr><td>K-by-P matrix</td></tr>
</table>

- *Q* is the length of the vector specified by the `ElevationAngles` property.
- *P* is the length of the vector specified by the `AzimuthAngles` property.
- *K* is the number of frequencies specified by the `FrequencyVector` property.

You can specify patterns for *L* targets by putting *L* patterns into a cell array. All patterns must have the same dimensions. The value of *L* must match the column dimensions of the signals passed as input into the function. However, you can use one pattern to model *L* targets.

RCS units are in square meters.

Example: `[1,1;1,1]`

Dependencies

To enable this property, set the `EnablePolarization` property to `false`.

Data Types: `double`

`ShhPattern` — Radar scattering matrix *HH* polarization component

`ones(181,361)` (default) | *Q*-by-*P* complex-valued matrix | *Q*-by-*P*-by-*K* complex-valued array | 1-by-*P*-by-*K* complex-valued array | *K*-by-*P* complex-valued matrix

Radar scattering matrix (SCM) *HH* polarization component, specified as a complex-valued matrix or array.

Dimensions	Application
Q-by-*P* matrix	Specifies the scattering matrix polarization component as a function of *Q* elevation angles and *P* azimuth angles. The same SCM matrix is used for all frequencies.
Q-by-*P*-by-*K* array	Specifies the scattering matrix polarization component as a function of *Q* elevation angles, *P* azimuth angles, and *K* frequencies. If *K* = *1*, the RCS pattern is equivalent to a *Q*-by-*P* matrix.
1-by-*P*-by-*K* array	Specifies the scattering matrix polarization component as a function of *P* azimuth angles and *K* frequencies. These dimension formats apply when there is only one elevation angle.
K-by-*P* matrix	

- *Q* is the length of the vector specified by the `ElevationAngles` property.
- *P* is the length of the vector specified by the `AzimuthAngles` property.
- *K* is the number of frequencies specified by the `FrequencyVector` property.

You can specify polarization component patterns for *L* targets by putting *L* patterns into a cell array. All patterns must have the same dimensions. The value of *L* must match the column dimensions of the signals passed as input into the function. You can, however, use one pattern to model *L* targets.

SCM units are in square-meters.

Example: `[1,1;1i,1i]`

Dependencies

To enable this property, set the `EnablePolarization` property to `true`.

Data Types: `double`
Complex Number Support: Yes

`SvvPattern` — Radar scattering matrix *VV* polarization component

`ones(181,361)` (default) | *Q*-by-*P* complex-valued matrix | *Q*-by-*P*-by-*K* complex-valued array | 1-by-*P*-by-*K* complex-valued array | *K*-by-*P* complex-valued matrix

Radar scattering matrix *VV-pol* component, specified as a complex-valued vector, matrix, or array. Different dimension cases have different applications.

Dimensions	Application
Q-by-*P* matrix	Specifies the scattering matrix polarization component as a function of *Q* elevation angles and *P* azimuth angles. The same SCM matrix is used for all frequencies.
Q-by-*P*-by-*K* array	Specifies the scattering matrix polarization component as a function of *Q* elevation angles, *P* azimuth angles, and *K* frequencies. If *K* = *1*, the RCS pattern is equivalent to a *Q*-by-*P* matrix.
1-by-*P*-by-*K* array	Specifies the scattering matrix polarization component as a function of *P* azimuth angles and

Dimensions	Application
K-by-*P* matrix	*K* frequencies. These dimension formats apply when there is only one elevation angle.

- *Q* is the length of the vector specified by the `ElevationAngles` property.
- *P* is the length of the vector specified by the `AzimuthAngles` property.
- *K* is the number of frequencies specified by the `FrequencyVector` property.

You can specify polarization component patterns for *L* targets by putting *L* patterns into a cell array. All patterns must have the same dimensions. The value of *L* must match the column dimensions of the signals passed as input into the function. You can, however, use one pattern to model *L* targets.

SCM units are in square-meters.

Example: `[1,1;1i,1i]`

Dependencies

To enable this property, set the `EnablePolarization` property to `true`.

Data Types: `double`
Complex Number Support: Yes

`ShvPattern` — Radar scattering matrix *HV* polarization component
`ones(181,361)` (default) | *Q*-by-*P* complex-valued matrix | *Q*-by-*P*-by-*K* complex-valued array | 1-by-*P*-by-*K* complex-valued vector | *K*-by-*P* complex-valued matrix

Radar scattering matrix (SCM) *HV-pol* component, specified as a complex-valued vector, matrix, or array. Different dimension cases have different applications.

Dimensions	Application
Q-by-*P* matrix	Specifies the scattering matrix polarization component as a function of *Q* elevation angles and *P* azimuth angles. The same SCM matrix is used for all frequencies.
Q-by-*P*-by-*K* array	Specifies the scattering matrix polarization component as a function of *Q* elevation angles, *P* azimuth angles, and *K* frequencies. If *K = 1*, the RCS pattern is equivalent to a *Q*-by-*P* matrix.
1-by-*P*-by-*K* array	Specifies the scattering matrix polarization component as a function of *P* azimuth angles and *K* frequencies. These dimension formats apply when there is only one elevation angle.
K-by-*P* matrix	

- *Q* is the length of the vector specified by the `ElevationAngles` property.
- *P* is the length of the vector specified by the `AzimuthAngles` property.
- *K* is the number of frequencies specified by the `FrequencyVector` property.

You can specify polarization component patterns for *L* targets by putting *L* patterns into a cell array. All patterns must have the same dimensions. The value of *L* must match the column dimensions of the signals passed as input into the function. You can, however, use one pattern to model *L* targets.

SCM units are in square-meters.

Example: [1,1;1i,1i]

Dependencies

To enable this property, set the EnablePolarization property to true.

Data Types: double
Complex Number Support: Yes

Model — Target fluctuation model
'Nonfluctuating' (default) | 'Swerling1' | 'Swerling2' | 'Swerling3' | 'Swerling4'

Target fluctuation model, specified as 'Nonfluctuating', 'Swerling1', 'Swerling2', 'Swerling3', or 'Swerling4'. If you set this property to a value other than 'Nonfluctuating', use the update input argument when calling the function.

Example: 'Swerling3'

Data Types: char

PropagationSpeed — Signal propagation speed
physconst('LightSpeed') (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by physconst('LightSpeed'). See physconst for more information.

Example: 3e8

Data Types: double

OperatingFrequency — Operating frequency
300e6 (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: 1e9

Data Types: double

SampleRate — Signal sample rate
1e6 (default) | positive real-valued scalar

Signal sample rate, specified as a positive real-valued scalar. Units are in hertz.

Example: 1e6

Data Types: double

NumSubbands — Number of processing subbands
64 (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: 128

Data Types: double

SeedSource — Seed source of random number generator for RCS fluctuation model
'Auto' (default) | 'Property'

Seed source of random number generator for RCS fluctuation model, specified as `'Auto'` or `'Property'`. When you set this property to:

- `'Auto'`, the object generates random numbers using the default MATLAB random number generator.
- `'Property'`, you specify the random number generator seed using the `Seed` property.

When using this object with Parallel Computing Toolbox software, set this property to `'Auto'`.

Dependencies

To enable this property, set the `Model` property to `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`.

Data Types: `string`

Seed — Random number generator seed
`0` (default) | nonnegative integer less than 2^{32}

Random number generator seed, specified as a nonnegative integer less than 2^{32}. .

Example: `32301`

Dependencies

To enable this property, set the `SeedSource` property to `'Property'`.

Data Types: `double`

Usage

Syntax

```
refl_sig = target(sig,ang)
refl_sig = target(sig,ang,update)

refl_sig = target(sig,ang,laxes)
refl_sig = target(sig,ang,laxes,update)
```

Description

`refl_sig = target(sig,ang)` returns the reflected signal, `refl_sig`, of an incident nonpolarized signal, `sig`. This syntax applies when you set the `EnablePolarization` property to `false` and the `Model` property to `'Nonfluctuating'`. In this case, the values specified in the `RCSPattern` property are used to compute the RCS values for the incident and reflected directions, `ang`.

`refl_sig = target(sig,ang,update)` uses `update` to control whether to update the RCS values. This syntax applies when you set the `EnablePolarization` property to `false` and the `Model` property to one of the fluctuating RCS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new RCS value is generated. If `update` is `false`, the previous RCS value is used.

`refl_sig = target(sig,ang,laxes)` returns the reflected signal, `refl_sig`, of an incident polarized signal, `sig`. This syntax applies when you set `EnablePolarization` to `true` and the

`Model` property to `'Nonfluctuating'`. The values specified in the `ShhPattern`, `SvvPattern`, and `ShvPattern` properties are used to compute the backscattering matrices for the incident directions, `ang`. The `laxes` argument specifies a local coordinate system used to define the horizontal and vertical polarization components.

`refl_sig = target(sig,ang,laxes,update)` uses the `update` argument to control whether to update the scattering matrix values. This syntax applies when you set the `EnablePolarization` property to `true` and the `Model` property to one of the fluctuating RCS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new RCS value is generated. If `update` is `false`, the previous RCS value is used.

Input Arguments

`sig` — Wideband signal

N-by-*M* complex-valued matrix | *1*-by-*M* `struct` array containing complex-valued fields

- Wideband nonpolarized signal, specified as an *N*-by-*M* complex-valued matrix. The quantity *N* is the number of signal samples and *M* is the number of independent signals reflecting off the target. Each column contains an independent signal reflected from the target.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
- Wideband polarized signal, specified as a 1-by-*M* `struct` array containing complex-valued fields. Each `struct` element contains three *N*-by-1 column vectors of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing the polarized signal that reflects from the target. Each `struct` element contains three *N*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

`ang` — Incident signal direction

2-by-1 real-valued column vector | 2-by-*M* real-valued matrix

Incident signal direction, specified as a real-valued column 2-by-1 vector or 2-by-*M* matrix. Each column of `ang` specifies the incident direction of the corresponding signal in the form `[AzimuthAngle;ElevationAngle]`. The number of columns in `ang` must match the number of independent signals in `sig`. Units are in degrees.

Example: `[30;45]`

Data Types: `double`

`update` — Update RCS

`false` (default) | `true`

Option to enable the RCS values for fluctuation models to update, specified as `false` or `true`. When `update` is `true`, a new RCS value is generated with each call to the function. If `update` is `false`, the RCS remains unchanged with each call to the function.

Data Types: `logical`

`laxes` — Local coordinate matrix
`eye(3,3)` (default) | 3-by-3 real-valued orthonormal matrix | 3-by-3-by-*M* real-valued array

Local coordinate system matrix, specified as a 3-by-3 real-valued orthonormal matrix or a 3-by-3-by-*M* real-valued array. The matrix columns specify the local coordinate system orthonormal *x*-axis, *y*-axis, and *z*-axis, respectively. Each axis is a vector of the form *(x;y;z)* with respect to the global coordinate system. When `sig` has only one signal, `laxes` is a 3-by-3 matrix. When `sig` has multiple signals, you can use a single 3-by-3 matrix for multiple signals in `sig`. In this case, all targets have the same local coordinate systems. When you specify `laxes` as a 3-by-3-by-*M* array, each page (third index) defines a 3-by-3 local coordinate matrix for the corresponding target.

Example: `[1,0,0;0,0.7071,-0.7071;0,0.7071,0.7071]`

Data Types: `double`

Output Arguments

`refl_sig` — Wideband reflected signal
N-by-*M* complex-valued matrix | *1*-by-*M* `struct` array containing complex-valued fields

- Wideband nonpolarized signal, returned as an *N*-by-*M* complex-valued matrix. Each column contains an independent signal reflected from the target.
- Wideband polarized signal, returned as a 1-by-*M* `struct` array containing complex-valued fields. Each `struct` element contains three *N*-by-1 column vectors of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing the polarized signal that reflects from the target.

The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to a reflecting angle.

For polarized fields, the `struct` element contains three *N*-by-1 complex-valued column vectors: `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

The output `refl_sig` contains signal samples arriving at the signal destination within the current input time frame. When the propagation time from source to destination exceeds the current time frame duration, the output does not contain all contributions from the input of the current time frame. The remaining output appears in the next call to the function.

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

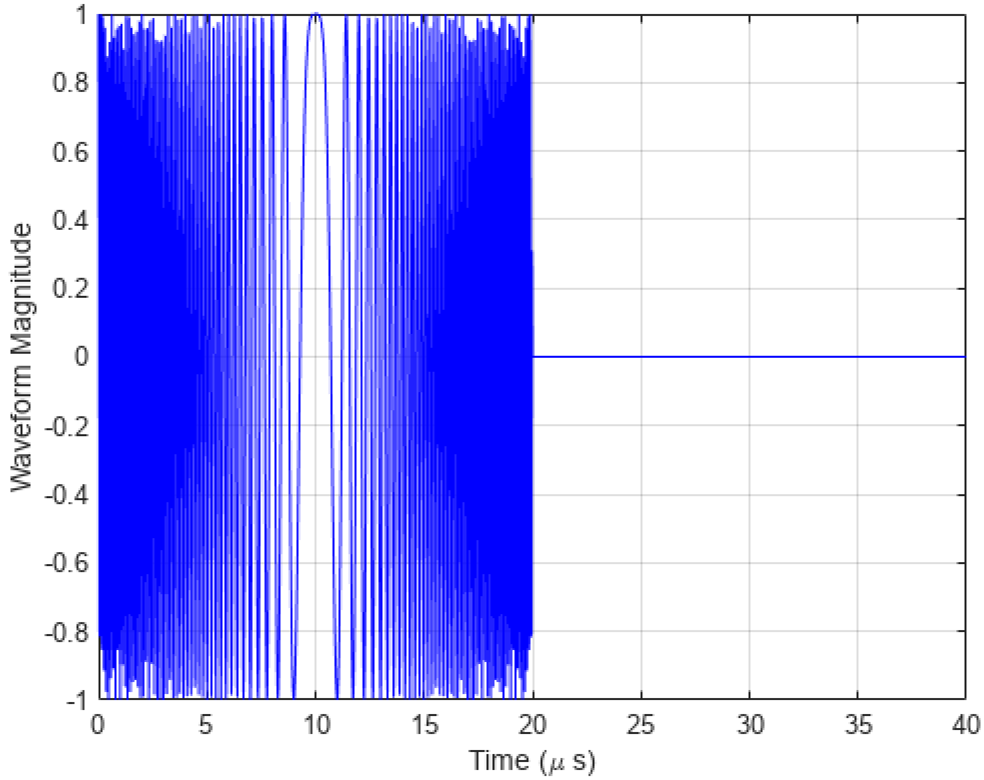
Examples

Backscatter Nonpolarized Wideband Signal

Calculate the reflected radar signal from a nonfluctuating point target having a peak RCS of 10.0 m^2. Use a simple target RCS pattern for illustrative purposes. Real RCS patterns are more complicated. The RCS pattern covers a range of angles from 10 – 30 degrees in azimuth and 5 – 15 degrees in elevation. The RCS peaks at 20 degrees azimuth and 10 degrees elevation. The RCS also has a frequency dependence and is specified at 5 frequencies within the signal bandwidth. Assume that the radar operating frequency is 100 MHz and that the signal is a linear FM waveform having a 20 MHz bandwidth.

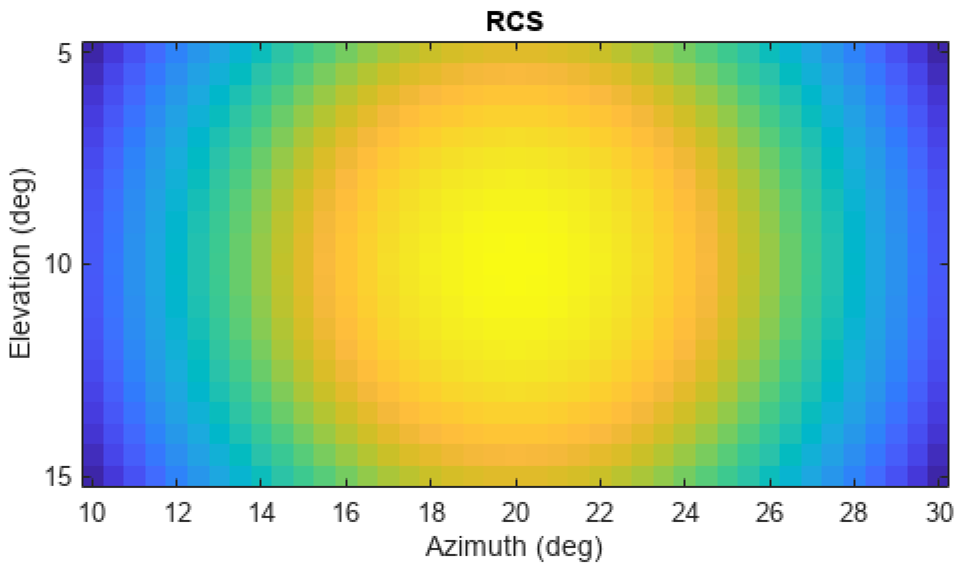
Create and plot the wideband signal.

```
c = physconst('LightSpeed');
fs = 50e6;
pw = 20e-6;
PRF = 1/(2*pw);
fc = 100e6;
bw = 20e6;
waveform = phased.LinearFMWaveform('SampleRate',fs,'PulseWidth',pw, ...
    'PRF',PRF,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',bw, ...
    'SweepDirection','Down','Envelope','Rectangular','SweepInterval', ...
    'Symmetric');
wav = waveform();
n = size(wav,1);
plot([0:(n-1)]/fs*1e6,real(wav),'b')
xlabel('Time (\mu s)')
ylabel('Waveform Magnitude')
grid
```



Create an RCS pattern at five different frequencies within the signal bandwidth using a simplified frequency dependence. The frequency dependence is unity at the operating frequency and falls off outside that frequency. Realistic frequency dependencies are more complicated. Plot the RCS pattern for one of the frequencies.

```
fvec = fc + [-fs/2,-fs/4,0,fs/4,fs/2];
fdep = cos(3*(1 - fvec/fc));
azmax = 20.0;
elmax = 10.0;
azpattern = [10.0:0.5:30.0];
elpattern = [5.0:0.5:15.0];
rcspattern0 = 10.0*cosd(4*(elpattern - elmax))'*cosd(4*(azpattern - azmax));
for k = 1:5
    rcspattern(:,:,k) = rcspattern0*fdep(k);
end
imagesc(azpattern,elpattern,abs(rcspattern(:,:,1)))
axis image
axis tight
title('RCS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
```

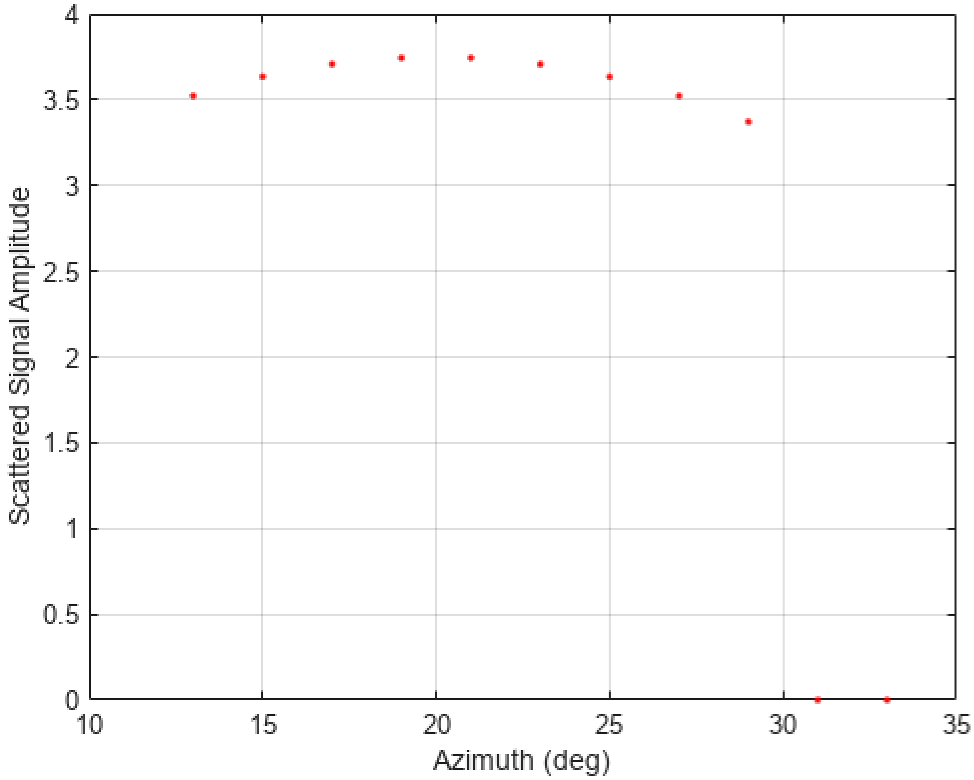


Create the `phased.WidebandBackscatterRadarTarget` System object™.

```
target = phased.WidebandBackscatterRadarTarget('Model','Nonfluctuating', ...
    'AzimuthAngles',azpattern,'ElevationAngles',elpattern,...
    'RCSPattern',rcspattern,'OperatingFrequency',fc,'NumSubbands',32, ...
    'FrequencyVector',fvec);
```

For a sequence of incident azimuth angles at constant elevation, find and plot the reflected signal amplitude.

```
az0 = 13.0;
el = 10.0;
az = az0 + [0:2:20];
naz = length(az);
magsig = zeros(1,naz);
for k = 1:naz
    y = target(wav,[az(k);el]);
    magsig(k) = max(abs(y));
end
plot(az,magsig,'r.')
xlabel('Azimuth (deg)')
ylabel('Scattered Signal Amplitude')
grid
```



Backscatter Nonpolarized Wideband Signal from Fluctuating Target

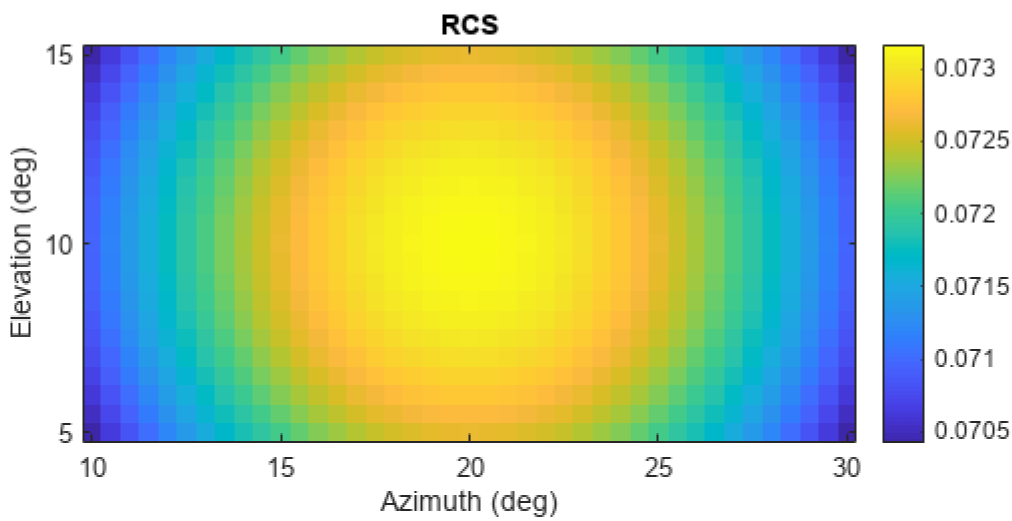
Calculate the reflected radar signal from a Swerling 4 fluctuating point target with a peak RCS of 0.1 m^2. Use a simple target RCS pattern for illustrative purposes. Real RCS patterns are more complicated. The RCS pattern covers a range of angles from 10 − 30 degrees in azimuth and 5 −15 degrees in elevation. The RCS peaks at 20 degrees in azimuth and 10 degrees in elevation at a value of 0.1 m^2. The RCS also has a frequency dependence and is specified at five frequencies within the signal bandwidth. Assume that the radar operating frequency is 100 MHz and that the signal is a linear FM waveform with a 20 MHz bandwidth. The sampling frequency is 50 MHz.

Create and plot the wideband signal.

```
c = physconst('LightSpeed');
fs = 50e6;
pw = 20e-6;
PRF = 1/(2*pw);
fc = 100.0e6;
bw = 20.0e6;
waveform = phased.LinearFMWaveform('SampleRate',fs,'PulseWidth',pw, ...
    'PRF',PRF,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',bw, ...
    'SweepDirection','Down','Envelope','Rectangular','SweepInterval', ...
    'Symmetric');
wav = waveform();
```

Create an RCS pattern at five different frequencies within the signal bandwidth using a simple frequency dependence. The frequency dependence is designed to be unity at the operating frequency and fall off outside that band. Realistic frequency dependencies are more complicated.

```
fvec = fc + [-fs/2,-fs/4,0,fs/4,fs/2];
fdep = cos(3*(1 - fvec/fc));
azmax = 20.0;
elmax = 10.0;
azangs = [10.0:0.5:30.0];
elangs = [5.0:0.5:15.0];
rcspattern0 = 0.1*(cosd((elangs - elmax))'*cosd((azangs - azmax))).^2;
for k = 1:5
    rcspattern(:,:,k) = rcspattern0*fdep(k);
end
imagesc(azangs,elangs,abs(rcspattern(:,:,5)))
axis image
axis xy
axis tight
title('RCS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
colorbar
```



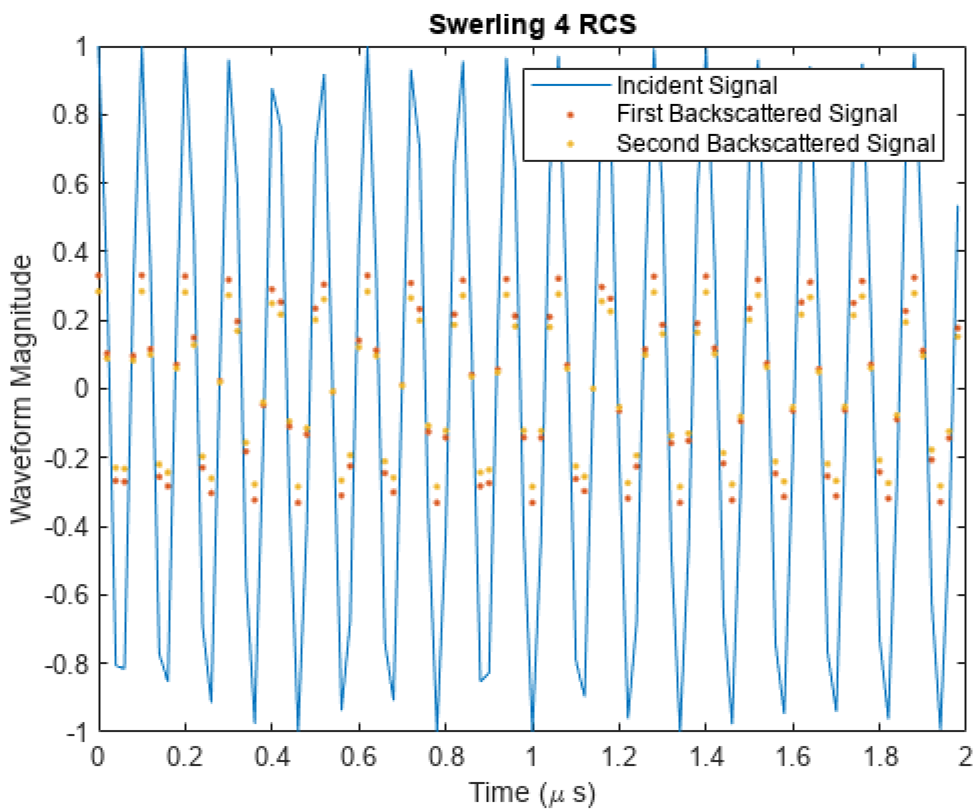
Create the `phased.WidebandBackscatterRadarTarget` System object™.

```
target = phased.WidebandBackscatterRadarTarget('Model','Swerling4', ...
    'SeedSource','Property','Seed',100213,'AzimuthAngles',azangs, ...
```

```
    'ElevationAngles',elangs,'RCSPattern',rcspattern, ...
    'OperatingFrequency',fc,'NumSubbands',32,'FrequencyVector',fvec);
```

Find and plot 100 samples of the incident signal and two sequential reflected signals at 10 degrees in azimuth and 10 degrees in elevation. Update the RCS at each execution of the System object™.

```
az = 10.0;
el = 10.0;
refl_wav1 = target(wav,[az;el],true);
refl_wav2 = target(wav,[az;el],true);
n = 100;
plot([0:(n-1)]/fs*1e6,real(wav(1:n)))
hold on
plot([0:(n-1)]/fs*1e6,real(refl_wav1(1:n)),'.')
plot([0:(n-1)]/fs*1e6,real(refl_wav2(1:n)),'.')
hold off
legend('Incident Signal','First Backscattered Signal','Second Backscattered Signal')
xlabel('Time (\mu s)')
ylabel('Waveform Magnitude')
title('Swerling 4 RCS')
```



More About

Backscattered Wideband Signals

Wideband signals are decomposed into narrowband signals which are reflected from the target independently.

For a narrowband nonpolarized signal, the reflected signal, Y, is

$$Y = \sqrt{G} \cdot X,$$

where:

- X is the incoming signal.
- G is the target gain factor, a dimensionless quantity given by

$$G = \frac{4\pi\sigma}{\lambda^2}.$$

  - σ is the mean radar cross-section (RCS) of the target.
  - λ is the wavelength of the incoming signal.

The incident signal on the target is scaled by the square root of the gain factor.

For narrowband polarized waves, the single scalar signal, X, is replaced by a vector signal, (E_H, E_V), with horizontal and vertical components. The scattering matrix, S, replaces the scalar cross-section, σ. Through the scattering matrix, the incident horizontal and vertical polarized signals are converted into the reflected horizontal and vertical polarized signals.

$$\begin{bmatrix} E_H^{(scat)} \\ E_V^{(scat)} \end{bmatrix} = \sqrt{\frac{4\pi}{\lambda^2}} \begin{bmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{bmatrix} \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix} = \sqrt{\frac{4\pi}{\lambda^2}} [S] \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

For further details, see [1] or [2].

Subband Frequency Processing

Subband processing decomposes a wideband signal into multiple subbands and applies narrowband processing to the signal in each subband. The signals for all subbands are summed to form the output signal.

When using wideband frequency System objects or blocks, you specify the number of subbands, N_B, in which to decompose the wideband signal. Subband center frequencies and widths are automatically computed from the total bandwidth and number of subbands. The total frequency band is centered on the carrier or operating frequency, f_c. The overall bandwidth is given by the sample rate, f_s. Frequency subband widths are $\Delta f = f_s/N_B$. The center frequencies of the subbands are

$$f_m = \begin{cases} f_c - \dfrac{f_s}{2} + (m-1)\Delta f, & N_B \text{ even} \\ f_c - \dfrac{(N_B - 1)f_s}{2N_B} + (m-1)\Delta f, & N_B \text{ odd} \end{cases}, \quad m = 1, \ldots, N_B$$

Some System objects let you obtain the subband center frequencies as output when you run the object. The returned subband frequencies are ordered consistently with the ordering of the discrete

Fourier transform. Frequencies above the carrier appear first, followed by frequencies below the carrier.

Version History

Introduced in R2016b

References

[1] Mott, H. *Antennas for Radar and Communications*. New York: John Wiley & Sons, 1992.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[3] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.RadarTarget` | `phased.BackscatterRadarTarget` | `phased.BackscatterSonarTarget` | `backscatterPedestrian`

Topics

“Modeling Target Radar Cross Section” (Radar Toolbox)
“Simulating Test Signals for a Radar Receiver”
“Swerling Target Models”

reset

System object: `phased.WidebandBackscatterRadarTarget`
Package: `phased`

Reset states of System object

Syntax

`reset(target)`

Description

`reset(target)` resets the internal state of the `phased.WidebandBackscatterRadarTarget` object, `target`. This method resets the random number generator state if `SeedSource` is a property of this System object and has the value `'Property'`.

Input Arguments

`target` — Wideband backscatter radar target
`phased.WidebandBackscatterRadarTarget` System object

Wideband backscatter radar target, specified as a `phased.WidebandBackscatterRadarTarget` System object.

Version History

Introduced in R2016b

step

System object: phased.WidebandBackscatterRadarTarget
Package: phased

Backscatter wideband signal from radar target

Syntax

```
refl_sig = step(target,sig,ang)
refl_sig = step(target,sig,ang,update)

refl_sig = step(target,sig,ang,laxes)
refl_sig = step(target,sig,ang,laxes,update)
```

Description

Note Alternatively, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`refl_sig = step(target,sig,ang)` returns the reflected signal, `refl_sig`, of an incident nonpolarized signal, `sig`. This syntax applies when you set the `EnablePolarization` property to `false` and the `Model` property to `'Nonfluctuating'`. In this case, the values specified in the `RCSPattern` property are used to compute the RCS values for the incident and reflected directions, `ang`.

`refl_sig = step(target,sig,ang,update)` uses `update` to control whether to update the RCS values. This syntax applies when you set the `EnablePolarization` property to `false` and the `Model` property to one of the fluctuating RCS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new RCS value is generated. If `update` is `false`, the previous RCS value is used.

`refl_sig = step(target,sig,ang,laxes)` returns the reflected signal, `refl_sig`, of an incident polarized signal, `sig`. This syntax applies when you set `EnablePolarization` to `true` and the `Model` property to `'Nonfluctuating'`. The values specified in the `ShhPattern`, `SvvPattern`, and `ShvPattern` properties are used to compute the backscattering matrices for the incident directions, `ang`. The `laxes` argument specifies a local coordinate system used to define the horizontal and vertical polarization components.

`refl_sig = step(target,sig,ang,laxes,update)` uses the `update` argument to control whether to update the scattering matrix values. This syntax applies when you set the `EnablePolarization` property to `true` and the `Model` property to one of the fluctuating RCS models: `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `update` is `true`, a new RCS value is generated. If `update` is `false`, the previous RCS value is used.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of

the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`target` — Wideband backscatter radar target
`phased.WidebandBackscatterRadarTarget` System object

Backscatter target, specified as a `phased.WidebandBackscatterRadarTarget` System object.

`sig` — Wideband signal
N-by-*M* complex-valued matrix | *1*-by-*M* `struct` array containing complex-valued fields

- Wideband nonpolarized signal, specified as an *N*-by-*M* complex-valued matrix. The quantity *N* is the number of signal samples and *M* is the number of independent signals reflecting off the target. Each column contains an independent signal reflected from the target.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
- Wideband polarized signal, specified as a 1-by-*M* `struct` array containing complex-valued fields. Each `struct` element contains three *N*-by-1 column vectors of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing the polarized signal that reflects from the target. Each `struct` element contains three *N*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

`ang` — Incident signal direction
2-by-1 real-valued column vector of positive values | 2-by-*M* real-valued matrix of positive values

Incident signal direction, specified as a real-valued column 2-by-1 vector or 2-by-*M* matrix of positive values. Each column of `ang` specifies the incident direction of the corresponding signal in the form `[AzimuthAngle;ElevationAngle]`. The number of columns in `ang` must match the number of independent signals in `sig`. Units are in degrees.

Example: `[30;45]`

Data Types: `double`

`update` — Update RCS
`false` (default) | `true`

Option to enable the RCS values for fluctuation models to update, specified as `false` or `true`. When `update` is `true`, a new RCS value is generated with each call to the `step` method. If `update` is `false`, the RCS remains unchanged with each call to `step`.

Data Types: `logical`

laxes — Local coordinate matrix
eye(3,3) (default) | 3-by-3 real-valued orthonormal matrix | 3-by-3-by-*M* real-valued array

Local coordinate system matrix, specified as a 3-by-3 real-valued orthonormal matrix or a 3-by-3-by-*M* real-valued array. The matrix columns specify the local coordinate system orthonormal *x*-axis, *y*-axis, and *z*-axis, respectively. Each axis is a vector of the form *(x;y;z)* with respect to the global coordinate system. When `sig` has only one signal, `laxes` is a 3-by-3 matrix. When `sig` has multiple signals, you can use a single 3-by-3 matrix for multiple signals in `sig`. In this case, all targets have the same local coordinate systems. When you specify `laxes` as a 3-by-3-by-*M* array, each page (third index) defines a 3-by-3 local coordinate matrix for the corresponding target.

Example: `[1,0,0;0,0.7071,-0.7071;0,0.7071,0.7071]`

Data Types: `double`

Output Arguments

refl_sig — Wideband reflected signal
N-by-*M* complex-valued matrix | *1*-by-*M* `struct` array containing complex-valued fields

- Wideband nonpolarized signal, returned as an *N*-by-*M* complex-valued matrix. Each column contains an independent signal reflected from the target.
- Wideband polarized signal, returned as a 1-by-*M* `struct` array containing complex-valued fields. Each `struct` element contains three *N*-by-1 column vectors of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing the polarized signal that reflects from the target.

The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to a reflecting angle.

For polarized fields, the `struct` element contains three *N*-by-1 complex-valued column vectors: `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

The output `refl_sig` contains signal samples arriving at the signal destination within the current input time frame. When the propagation time from source to destination exceeds the current time frame duration, the output does not contain all contributions from the input of the current time frame. The remaining output appears in the next call to `step`.

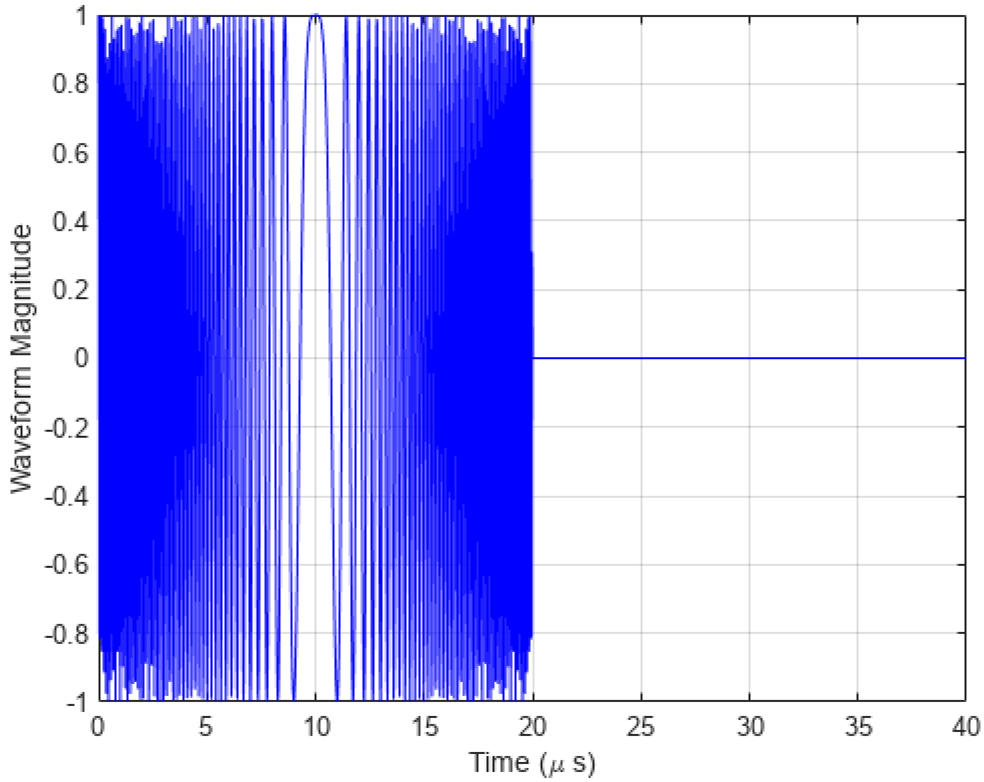
Examples

Backscatter Nonpolarized Wideband Signal

Calculate the reflected radar signal from a nonfluctuating point target having a peak RCS of 10.0 m^2. Use a simple target RCS pattern for illustrative purposes. Real RCS patterns are more complicated. The RCS pattern covers a range of angles from 10 – 30 degrees in azimuth and 5 – 15 degrees in elevation. The RCS peaks at 20 degrees azimuth and 10 degrees elevation. The RCS also has a frequency dependence and is specified at 5 frequencies within the signal bandwidth. Assume that the radar operating frequency is 100 MHz and that the signal is a linear FM waveform having a 20 MHz bandwidth.

Create and plot the wideband signal.

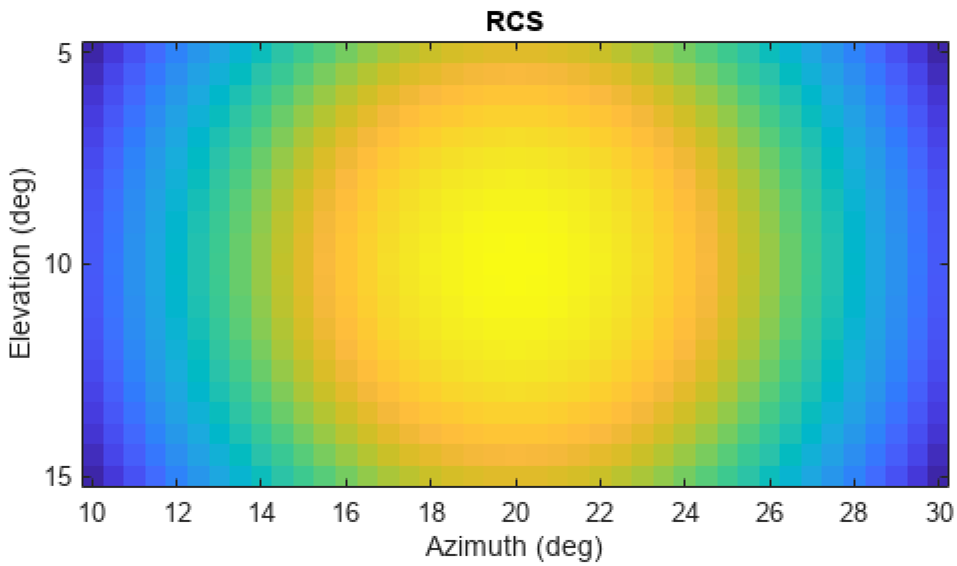
```
c = physconst('LightSpeed');
fs = 50e6;
pw = 20e-6;
PRF = 1/(2*pw);
fc = 100e6;
bw = 20e6;
waveform = phased.LinearFMWaveform('SampleRate',fs,'PulseWidth',pw, ...
    'PRF',PRF,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',bw, ...
    'SweepDirection','Down','Envelope','Rectangular','SweepInterval', ...
    'Symmetric');
wav = waveform();
n = size(wav,1);
plot([0:(n-1)]/fs*1e6,real(wav),'b')
xlabel('Time (\mu s)')
ylabel('Waveform Magnitude')
grid
```



Create an RCS pattern at five different frequencies within the signal bandwidth using a simplified frequency dependence. The frequency dependence is unity at the operating frequency and falls off outside that frequency. Realistic frequency dependencies are more complicated. Plot the RCS pattern for one of the frequencies.

```
fvec = fc + [-fs/2,-fs/4,0,fs/4,fs/2];
fdep = cos(3*(1 - fvec/fc));
azmax = 20.0;
elmax = 10.0;
azpattern = [10.0:0.5:30.0];
```

```
elpattern = [5.0:0.5:15.0];
rcspattern0 = 10.0*cosd(4*(elpattern - elmax))'*cosd(4*(azpattern - azmax));
for k = 1:5
    rcspattern(:,:,k) = rcspattern0*fdep(k);
end
imagesc(azpattern,elpattern,abs(rcspattern(:,:,1)))
axis image
axis tight
title('RCS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
```



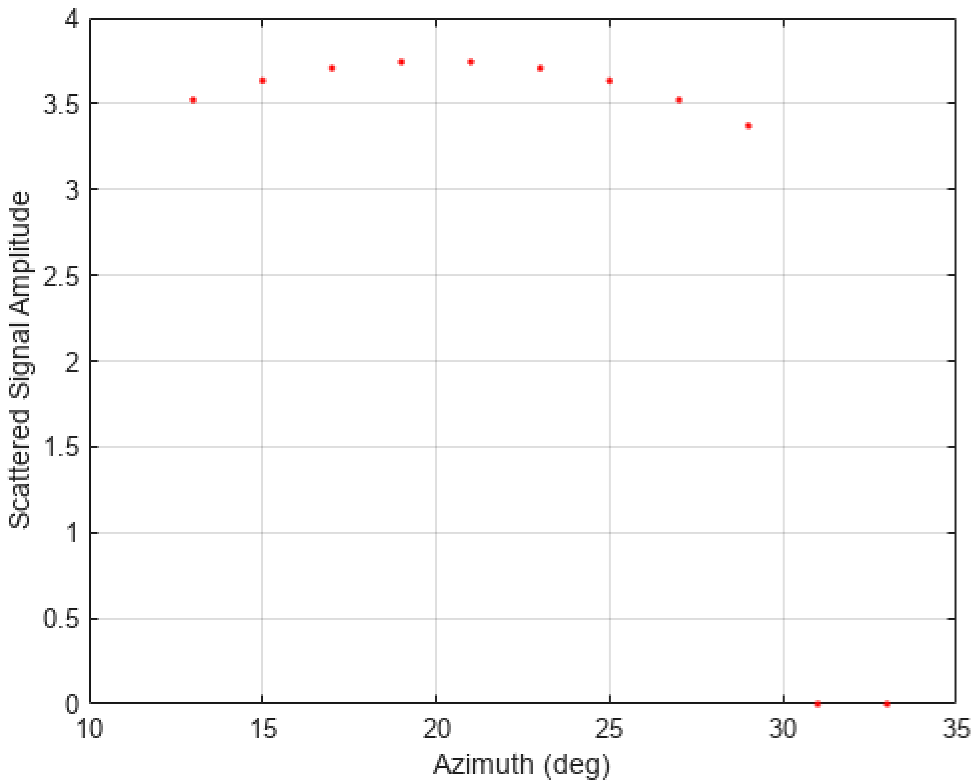
Create the `phased.WidebandBackscatterRadarTarget` System object™.

```
target = phased.WidebandBackscatterRadarTarget('Model','Nonfluctuating', ...
    'AzimuthAngles',azpattern,'ElevationAngles',elpattern,...
    'RCSPattern',rcspattern,'OperatingFrequency',fc,'NumSubbands',32, ...
    'FrequencyVector',fvec);
```

For a sequence of incident azimuth angles at constant elevation, find and plot the reflected signal amplitude.

```
az0 = 13.0;
el = 10.0;
az = az0 + [0:2:20];
naz = length(az);
magsig = zeros(1,naz);
for k = 1:naz
```

```
    y = target(wav,[az(k);el]);
    magsig(k) = max(abs(y));
end
plot(az,magsig,'r.')
xlabel('Azimuth (deg)')
ylabel('Scattered Signal Amplitude')
grid
```



Backscatter Nonpolarized Wideband Signal from Fluctuating Target

Calculate the reflected radar signal from a Swerling 4 fluctuating point target with a peak RCS of 0.1 m^2. Use a simple target RCS pattern for illustrative purposes. Real RCS patterns are more complicated. The RCS pattern covers a range of angles from 10 − 30 degrees in azimuth and 5 −15 degrees in elevation. The RCS peaks at 20 degrees in azimuth and 10 degrees in elevation at a value of 0.1 m^2. The RCS also has a frequency dependence and is specified at five frequencies within the signal bandwidth. Assume that the radar operating frequency is 100 MHz and that the signal is a linear FM waveform with a 20 MHz bandwidth. The sampling frequency is 50 MHz.

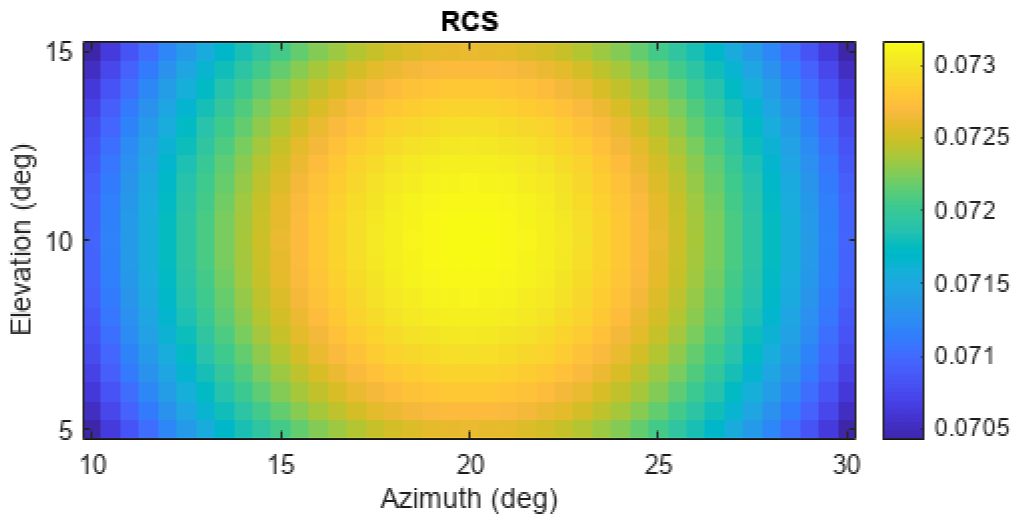
Create and plot the wideband signal.

```
c = physconst('LightSpeed');
fs = 50e6;
pw = 20e-6;
PRF = 1/(2*pw);
```

```
fc = 100.0e6;
bw = 20.0e6;
waveform = phased.LinearFMWaveform('SampleRate',fs,'PulseWidth',pw, ...
    'PRF',PRF,'OutputFormat','Pulses','NumPulses',1,'SweepBandwidth',bw, ...
    'SweepDirection','Down','Envelope','Rectangular','SweepInterval', ...
    'Symmetric');
wav = waveform();
```

Create an RCS pattern at five different frequencies within the signal bandwidth using a simple frequency dependence. The frequency dependence is designed to be unity at the operating frequency and fall off outside that band. Realistic frequency dependencies are more complicated.

```
fvec = fc + [-fs/2,-fs/4,0,fs/4,fs/2];
fdep = cos(3*(1 - fvec/fc));
azmax = 20.0;
elmax = 10.0;
azangs = [10.0:0.5:30.0];
elangs = [5.0:0.5:15.0];
rcspattern0 = 0.1*(cosd((elangs - elmax))'*cosd((azangs - azmax))).^2;
for k = 1:5
    rcspattern(:,:,k) = rcspattern0*fdep(k);
end
imagesc(azangs,elangs,abs(rcspattern(:,:,5)))
axis image
axis xy
axis tight
title('RCS')
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
colorbar
```

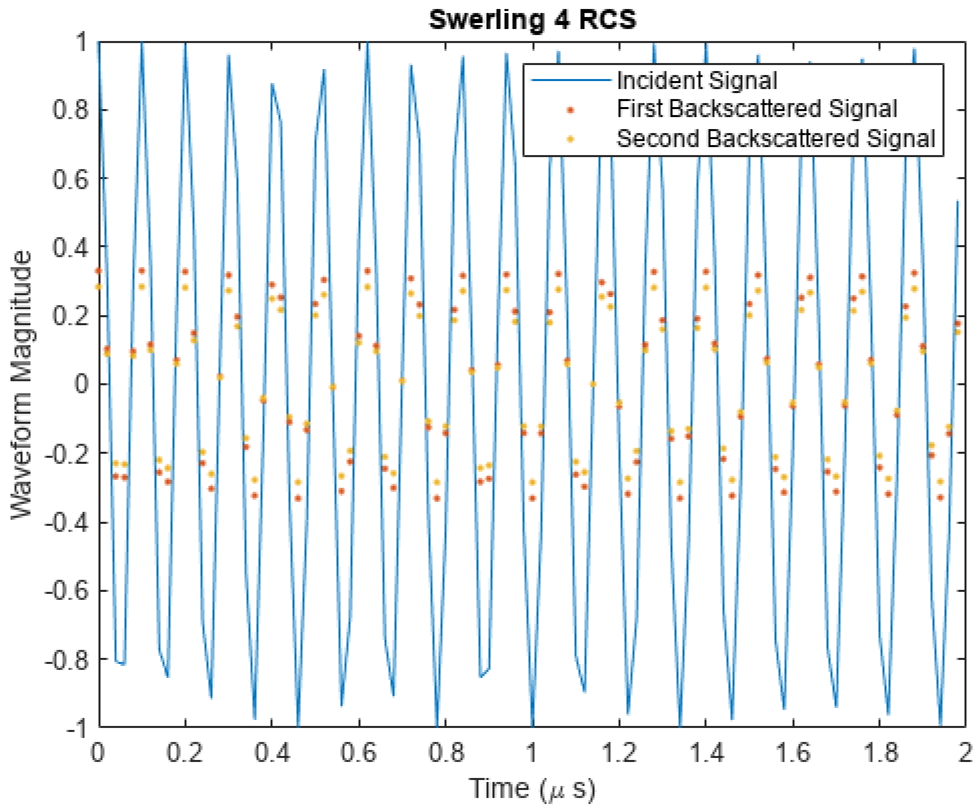


Create the `phased.WidebandBackscatterRadarTarget` System object™.

```
target = phased.WidebandBackscatterRadarTarget('Model','Swerling4', ...
    'SeedSource','Property','Seed',100213,'AzimuthAngles',azangs, ...
    'ElevationAngles',elangs,'RCSPattern',rcspattern, ...
    'OperatingFrequency',fc,'NumSubbands',32,'FrequencyVector',fvec);
```

Find and plot 100 samples of the incident signal and two sequential reflected signals at 10 degrees in azimuth and 10 degrees in elevation. Update the RCS at each execution of the System object™.

```
az = 10.0;
el = 10.0;
refl_wav1 = target(wav,[az;el],true);
refl_wav2 = target(wav,[az;el],true);
n = 100;
plot([0:(n-1)]/fs*1e6,real(wav(1:n)))
hold on
plot([0:(n-1)]/fs*1e6,real(refl_wav1(1:n)),'.')
plot([0:(n-1)]/fs*1e6,real(refl_wav2(1:n)),'.')
hold off
legend('Incident Signal','First Backscattered Signal','Second Backscattered Signal')
xlabel('Time (\mu s)')
ylabel('Waveform Magnitude')
title('Swerling 4 RCS')
```



Version History

Introduced in R2016b

phased.WidebandCollector

Package: `phased`

Wideband signal collector

Description

The `phased.WidebandCollector` System object implements a wideband signal collector. A collector converts incident wideband wave fields arriving from specified directions into signals to be further processed. Wave fields are incident on antenna and microphone elements, sensor arrays, or subarrays. The object collects signals in one of two ways controlled by the `Wavefront` Wavefront property.

- If the Wavefront property is set to `'Plane'`, the collected signals at each element or subarray are the coherent sum of all incident plane wave fields sampled at each array element or subarray.
- If the Wavefront property is set to `'Unspecified'`, the collected signals are formed from an independent field incident on each individual sensor element.

You can use this object to

- model arriving signals as polarized or non-polarized fields depending upon whether the element or array supports polarization and the value of the Polarization property. Using polarization, you can receive a signal as a polarized electromagnetic field, or receive two independent signals using orthogonal polarization directions.
- model acoustic fields by using nonpolarized microphone and sonar transducer array elements and by setting the “Polarization” on page 1-0 to `'None'`. You must also set the PropagationSpeed to a value appropriate for the medium.
- collect fields at subarrays created by the `phased.ReplicatedSubarray` and `phased.PartitionedArray` objects. You can steer all subarrays in the same direction using the steering angle argument, `STEERANG`, or steer each subarray in a different direction using the subarray element weights argument, `WS`. You cannot set the Wavefront property to `'Unspecified'` for subarrays.

To collect arriving signals at the elements or arrays:

1. Create the `phased.WidebandCollector` object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
collector = phased.WidebandCollector
collector = phased.WidebandCollector(Name,Value)
```

Description

`collector = phased.WidebandCollector` creates a wideband signal collector object, `collector`, with default property values.

`collector = phased.WidebandCollector(Name,Value)` creates a wideband signal collector with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `collector = phased.WidebandCollector('Sensor',phased.URA,'CarrierFrequency',300e6)` sets the sensor array to a uniform rectangular array (URA) with default URA property values. The beamformer assumes a carrier frequency of 300 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Sensor` — Sensor element or sensor array
`phased.ULA` array with default property values (default) | Phased Array System Toolbox sensor or array

Sensor element or sensor array, specified as a System object belonging to Phased Array System Toolbox. A sensor array can contain subarrays.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`SampleRate` — Signal sample rate
`1e6` (default) | positive real-valued scalar

Signal sample rate, specified as a positive real-valued scalar. Units are in hertz.

Example: `1e6`

Data Types: `double`

`ModulatedInput` — Assume modulated input
`true` (default) | `false`

Set this property to `true` to indicate the input signal is demodulated at a carrier frequency.

Data Types: `logical`

`CarrierFrequency` — Carrier frequency
`1e9` (default) | positive real-valued scalar

Carrier frequency, specified as a positive real-valued scalar. Units are in hertz.

Example: `1e6`

Data Types: `double`

`NumSubbands` — Number of processing subbands
`64` (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: `128`

Data Types: `double`

`SensorGainMeasure` — Specify sensor gain
`'dB'` (default) | `'dBi'`

Sensor gain measure, specified as `'dB'` or `'dBi'`.

- When you set this property to `'dB'`, the input signal power is scaled by the sensor power pattern (in dB) at the corresponding direction and then combined.
- When you set this property to `'dBi'`, the input signal power is scaled by the directivity pattern (in dBi) at the corresponding direction and then combined. This option is useful when you want to compare results with the values predicted by the radar equation that uses dBi to specify the antenna gain. The computation using the `'dBi'` option is expensive as it requires an integration over all directions to compute the total radiated power of the sensor.

Data Types: `char`

`Wavefront` — Type of incoming wavefront
`'Plane'` (default) | `'Unspecified'`

The type of incoming wavefront, specified as `'Plane'` or `'Unspecified'`:

- `'Plane'` — input signals are multiple plane waves impinging on the entire array. Each plane wave is received by all collecting elements.
- `'Unspecified'` — collected signals are independent fields incident on individual sensor elements. If the `Sensor` property is an array that contains subarrays, you cannot set the `Wavefront` property to `'Unspecified'`.

Data Types: `char`

`Polarization` — Polarization configuration
`'None'` (default) | `'Combined'` | `'Dual'`

Polarization configuration, specified as `'None'`, `'Combined'`, or `'Dual'`. When you set this property to `'None'`, the incident fields are considered scalar fields. When you set this property to `'Combined'`, the incident fields are polarized and represent a single arriving signal whose polarization reflects the sensor's inherent polarization. When you set this property to `'Dual'`, the *H* and *V* polarization components of the fields are independent signals.

Example: `'Dual'`

Data Types: `char`

`WeightsInputPort` — Enable weights input
`false` (default) | `true`

Enable weights input, specified as `false` or `true`. When `true`, use the object input argument `W` to specify weights. Weights are applied to individual array elements (or at the subarray level when subarrays are supported).

Data Types: `logical`

Usage

Syntax

```
Y = collector(X,ANG)
Y = collector(X,ANG,LAXES)
[YH,YV] = collector(X,ANG,LAXES)
[___] = collector(___,W)
[___] = collector(___,STEERANG)
[___] = collector(___,WS)
```

Description

`Y = collector(X,ANG)` collects the signals, `X`, arriving from the directions specified by `ANG`. `Y` contains the collected signals.

`Y = collector(X,ANG,LAXES)` also specifies `LAXES` as the local coordinate system axes directions. To use this syntax, set the property to `'Combined'`.

`[YH,YV] = collector(X,ANG,LAXES)` returns an H-polarization component of the field, `YH`, and a V-polarization component, `YV`. To use this syntax, set the Polarization property to `'Dual'`.

`[___] = collector(___,W)` also specifies `W` as array element or subarray weights. To use this syntax, set the WeightsInputPort property to `true`.

`[___] = collector(___,STEERANG)` also specifies `STEERANG` as the subarray steering angle. To use this syntax, set the Sensor property to an array that supports subarrays and set the `SubarraySteering` property of that array to either `'Phase'` or `'Time'`.

`[___] = collector(___,WS)` also specifies `WS` as the weights applied to each element within each subarray. To use this syntax, set the Sensor property to an array that supports subarrays and set the `SubarraySteering` of that array to `'Custom'`.

Input Arguments

`X` — Arriving signals
complex-valued *M*-by-*L* matrix | complex-valued 1-by-*L* cell array of structures

Arriving signals, specified as a complex-valued *M*-by-*L* matrix or complex-valued 1-by-*L* cell array of structures. *M* is the number of signal samples and *L* is the number of arrival angles. This argument represents the arriving fields.

- If the `Polarization` property value is set to `'None'`, X is an *M*-by-*L* matrix.
- If the `Polarization` property value is set to `'Combined'` or `'Dual'`, X is a 1-by-*L* cell array of structures. Each cell corresponds to a separate arriving signal. Each `struct` contains three column vectors containing the *X*, *Y*, and *Z* components of the polarized fields defined with respect to the global coordinate system.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this argument, set the `Polarization` property to `'None'` or `'Combined'`.

Data Types: `double`
Complex Number Support: Yes

ANG — Arrival directions of signals
real-valued 2-by-*L* matrix

Arrival directions of signals, specified as a real-valued 2-by-*L* matrix. Each column specifies an arrival direction in the form `[AzimuthAngle;ElevationAngle]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive. When the Wavefront property is `false`, the number of angles must equal the number of array elements, *N*. Units are in degrees.

Example: `[30,20;45,0]`

Data Types: `double`

LAXES — Local coordinate system
real-valued 3-by-3 orthogonal matrix

Local coordinate system, specified as a real-valued 3-by-3 orthogonal matrix. The matrix columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes with respect to the global coordinate system.

Example: `rotx(30)`

Dependencies

To enable this argument, set the `Polarization` property to `'Combined'` or `'Dual'`.

Data Types: `double`

W — Element or subarray weights
N-by-1 column vector

Element or subarray weights, specified as a complex-valued *N*-by-1 column vector where *N* is the number of array elements (or subarrays when the array supports subarrays).

Dependencies

To enable this argument, set the WeightsInputPort property to `true`.

Data Types: `double`
Complex Number Support: Yes

`WS` — Subarray element weights
complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray element weights

Sensor Array	Subarray weights
`phased.ReplicatedSubarray`	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of `WS` specifies the weights for the corresponding subarray.
`phased.PartitionedArray`	Subarrays may not have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `Sensor` property to an array that contains subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

`STEERANG` — Subarray steering angle
real-valued 2-by-1 vector

Subarray steering angle, specified as a length-2 column vector. The vector has the form `[azimuthAngle;elevationAngle]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive. Units are in degrees.

Example: `[20;15]`

Dependencies

To enable this argument, set the `Sensor` property to an array that supports subarrays and set the `SubarraySteering` property of that array to either `'Phase'` or `'Time'`

Data Types: `double`

Output Arguments

Y — Collected signal
complex-valued *M*-by-*N* matrix

Collected signal, returned as a complex-valued *M*-by-*N* matrix. *M* is the length of the input signal. *N* is the number of array elements (or subarrays when subarrays are supported). Each column corresponds to the signal collected by the corresponding array element (or corresponding subarrays when subarrays are supported).

Dependencies

To enable this argument, set the `Polarization` property to `'None'` or `'Combined'`.

Data Types: `double`

YH — Collected horizontal polarization signal
complex-valued *M*-by-*N* matrix

Collected horizontal polarization signal, returned as a complex-valued *M*-by-*N* matrix. *M* is the length of the input signal. *N* is the number of array elements (or subarrays when subarrays are supported). Each column corresponds to the signal collected by the corresponding array element (or corresponding subarrays when subarrays are supported).

Dependencies

To enable this argument, set the `Polarization` property to `'Dual'`.

Data Types: `double`

YV — Collected vertical polarization signal
complex-valued *M*-by-*N* matrix

Collected horizontal polarization signal, returned as a complex-valued *M*-by-*N* matrix. *M* is the length of the input signal. *N* is the number of array elements (or subarrays when subarrays are supported). Each column corresponds to the signal collected by the corresponding array element (or corresponding subarrays when subarrays are supported).

Dependencies

To enable this argument, set the `Polarization` property to `'Dual'`.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step   Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Collect Wideband Signal at Single Antenna

Use the `phased.WidebandCollector` System object™ to construct a signal arriving at a single isotropic antenna from 10° azimuth and 30° elevation.

```
antenna = phased.IsotropicAntennaElement;
collector = phased.WidebandCollector('Sensor',antenna);
x = [1;0;-1];
incidentAngle = [10;30];
y = collector(x,incidentAngle);
disp(y)
```

```
   1.0000 + 0.0000i
   0.0000 + 0.0000i
  -1.0000 - 0.0000i
```

Collect Wideband Signal at 5-Element ULA

Use the wideband collector to construct the signal impinging upon a 5-element ULA of isotropic antennas from 10 degrees azimuth and 30 degrees elevation.

```
array = phased.ULA('NumElements',5);
collector = phased.WidebandCollector('Sensor',array);
x = [1;1;1];
incidentAngle = [10;30];
y = collector(x,incidentAngle);
disp(y)
```

```
  -0.9997 + 0.0102i  -0.0051 - 0.9999i   1.0000 + 0.0000i  -0.0051 + 1.0001i  -1.0002 - 0.0102i
  -0.9999 + 0.0102i  -0.0051 - 1.0000i   1.0000 + 0.0000i  -0.0051 + 1.0000i  -0.9999 - 0.0102i
  -1.0002 + 0.0102i  -0.0051 - 1.0001i   1.0000 - 0.0000i  -0.0051 + 0.9999i  -0.9997 - 0.0102i
```

Collect Different Signals at 3-Element ULA

Collect three signals incoming into a 3-element array of isotropic antenna elements. Each antenna collects a separate input signal from a separate direction.

```
array = phased.ULA('NumElements',3);
collector = phased.WidebandCollector('Sensor',array,...
    'Wavefront','Unspecified');
rng default
x = rand(10,3);
incidentAngles = [10 20 45; 0 5 2];
y = collector(x,incidentAngles);
disp(y)
```

```
0.8147 + 0.0000i   0.1576 + 0.0000i   0.6557 - 0.0000i
0.9058 + 0.0000i   0.9706 + 0.0000i   0.0357 + 0.0000i
0.1270 - 0.0000i   0.9572 + 0.0000i   0.8491 - 0.0000i
0.9134 - 0.0000i   0.4854 + 0.0000i   0.9340 - 0.0000i
0.6324 + 0.0000i   0.8003 + 0.0000i   0.6787 - 0.0000i
0.0975 - 0.0000i   0.1419 + 0.0000i   0.7577 - 0.0000i
0.2785 + 0.0000i   0.4218 - 0.0000i   0.7431 - 0.0000i
0.5469 - 0.0000i   0.9157 - 0.0000i   0.3922 - 0.0000i
0.9575 + 0.0000i   0.7922 + 0.0000i   0.6555 + 0.0000i
0.9649 + 0.0000i   0.9595 + 0.0000i   0.1712 - 0.0000i
```

More About

Subband Frequency Processing

Subband processing decomposes a wideband signal into multiple subbands and applies narrowband processing to the signal in each subband. The signals for all subbands are summed to form the output signal.

When using wideband frequency System objects or blocks, you specify the number of subbands, N_B, in which to decompose the wideband signal. Subband center frequencies and widths are automatically computed from the total bandwidth and number of subbands. The total frequency band is centered on the carrier or operating frequency, f_c. The overall bandwidth is given by the sample rate, f_s. Frequency subband widths are $\Delta f = f_s/N_B$. The center frequencies of the subbands are

$$f_m = \begin{cases} f_c - \dfrac{f_s}{2} + (m-1)\Delta f, & N_B \text{ even} \\ f_c - \dfrac{(N_B - 1)f_s}{2N_B} + (m-1)\Delta f, & N_B \text{ odd} \end{cases}, \quad m = 1, \ldots, N_B$$

Some System objects let you obtain the subband center frequencies as output when you run the object. The returned subband frequencies are ordered consistently with the ordering of the discrete Fourier transform. Frequencies above the carrier appear first, followed by frequencies below the carrier.

The `phased.WidebandCollector` System object uses the narrowband phased approximation of the time delays across receiving elements in the far field for each subband.

Algorithms

If the `Wavefront` property value is `'Plane'`, `phased.WidebandCollector` does the following for each plane wave signal:

1. Decomposes the signal into multiple subbands.
2. Uses the phase approximation of the time delays across collecting elements in the far field for each subband.
3. Regroups the collected signals in all the subbands to form the output signal.

If the `Wavefront` property value is `'Unspecified'`, the object collects each channel independently.

For further details, see [1].

Version History

Introduced in R2011a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Requires dynamic memory allocation. See “Limitations for System Objects that Require Dynamic Memory Allocation”.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`phased.Collector` | `phased.WidebandRadiator` | `phased.Radiator`

step

System object: `phased.WidebandCollector`
Package: `phased`

Collect signals

Syntax

```
Y = step(H,X,ANG)
Y = step(H,X,ANG,LAXES)
Y = step(H,X,ANG,WEIGHTS)
Y = step(H,X,ANG,STEERANGLE)
Y = step(H,X,ANG,LAXES,WEIGHTS,STEERANGLE)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`Y = step(H,X,ANG)` collects signals `X` arriving from directions `ANG`. The collection process depends on the `Wavefront` property of `H`, as follows:

- If `Wavefront` has the value `'Plane'`, each collecting element collects all the far field signals in `X`. Each column of `Y` contains the output of the corresponding element in response to all the signals in `X`.
- If `Wavefront` has the value `'Unspecified'`, each collecting element collects only one impinging signal from `X`. Each column of `Y` contains the output of the corresponding element in response to the corresponding column of `X`. The `'Unspecified'` option is available when the `Sensor` property of `H` does not contain subarrays.

`Y = step(H,X,ANG,LAXES)` uses `LAXES` as the local coordinate system axes directions. This syntax is available when you set the `EnablePolarization` property to `true`.

`Y = step(H,X,ANG,WEIGHTS)` uses `WEIGHTS` as the weight vector. This syntax is available when you set the `WeightsInputPort` property to `true`.

`Y = step(H,X,ANG,STEERANGLE)` uses `STEERANGLE` as the subarray steering angle. This syntax is available when you configure `H` so that `H.Sensor` is an array that contains subarrays and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

`Y = step(H,X,ANG,LAXES,WEIGHTS,STEERANGLE)` combines all input arguments. This syntax is available when you configure `H` so that `H.WeightsInputPort` is `true`, `H.Sensor` is an array that contains subarrays, and `H.Sensor.SubarraySteering` is either `'Phase'` or `'Time'`.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of

the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

H

Collector object.

X

Arriving signals. Each column of X represents a separate signal. The specific interpretation of X depends on the `Wavefront` property of H.

Wavefront Property Value	Description
`'Plane'`	Each column of X is a far field signal.
`'Unspecified'`	Each column of X is the signal impinging on the corresponding element. In this case, the number of columns in X must equal the number of collecting elements in the `Sensor` property.

- If the `EnablePolarization` property value is set to `false`, X is a matrix. The number of columns of the matrix equals the number of separate signals.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
- If the `EnablePolarization` property value is set to `true`, X is a row vector of MATLAB `struct` type. The dimension of the `struct` array equals the number of separate signals. Each `struct` member contains three column-vector fields, X, Y, and Z, representing the *x*, *y*, and *z* components of the polarized wave vector signals in the global coordinate system.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

ANG

Incident directions of signals, specified as a two-row matrix. Each column specifies the incident direction of the corresponding column of X. Each column of ANG has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

LAXES

Local coordinate system. LAXES is a 3-by-3 matrix whose columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively. Each axis is specified in terms of `[x;y;z]` with respect to the global coordinate system. This argument is only used when the `EnablePolarization` property is set to `true`.

WEIGHTS

Vector of weights. WEIGHTS is a column vector of length M, where M is the number of collecting elements.

Default: ones(M,1)

STEERANGLE

Subarray steering angle, specified as a length-2 column vector. The vector has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

Output Arguments

Y

Collected signals. Each column of Y contains the output of the corresponding element. The output is the response to all the signals in X, or one signal in X, depending on the Wavefront property of H.

Examples

Collect Wideband Signal at Single Antenna

Use the phased.WidebandCollector System object™ to construct a signal arriving at a single isotropic antenna from 10° azimuth and 30° elevation.

```
antenna = phased.IsotropicAntennaElement;
collector = phased.WidebandCollector('Sensor',antenna);
x = [1;0;-1];
incidentAngle = [10;30];
y = collector(x,incidentAngle);
disp(y)
```

```
   1.0000 + 0.0000i
   0.0000 + 0.0000i
  -1.0000 - 0.0000i
```

Collect Wideband Signal at 5-Element ULA

Use the wideband collector to construct the signal impinging upon a 5-element ULA of isotropic antennas from 10 degrees azimuth and 30 degrees elevation.

```
array = phased.ULA('NumElements',5);
collector = phased.WidebandCollector('Sensor',array);
x = [1;1;1];
incidentAngle = [10;30];
y = collector(x,incidentAngle);
disp(y)
```

```
-0.9997 + 0.0102i  -0.0051 - 0.9999i   1.0000 + 0.0000i  -0.0051 + 1.0001i  -1.0002 - 0.0102i
-0.9999 + 0.0102i  -0.0051 - 1.0000i   1.0000 + 0.0000i  -0.0051 + 1.0000i  -0.9999 - 0.0102i
-1.0002 + 0.0102i  -0.0051 - 1.0001i   1.0000 - 0.0000i  -0.0051 + 0.9999i  -0.9997 - 0.0102i
```

Collect Different Signals at 3-Element ULA

Collect three signals incoming into a 3-element array of isotropic antenna elements. Each antenna collects a separate input signal from a separate direction.

```
array = phased.ULA('NumElements',3);
collector = phased.WidebandCollector('Sensor',array,...
    'Wavefront','Unspecified');
rng default
x = rand(10,3);
incidentAngles = [10 20 45; 0 5 2];
y = collector(x,incidentAngles);
disp(y)
```

```
   0.8147 + 0.0000i   0.1576 + 0.0000i   0.6557 - 0.0000i
   0.9058 + 0.0000i   0.9706 + 0.0000i   0.0357 + 0.0000i
   0.1270 - 0.0000i   0.9572 + 0.0000i   0.8491 - 0.0000i
   0.9134 - 0.0000i   0.4854 + 0.0000i   0.9340 - 0.0000i
   0.6324 + 0.0000i   0.8003 + 0.0000i   0.6787 - 0.0000i
   0.0975 - 0.0000i   0.1419 + 0.0000i   0.7577 - 0.0000i
   0.2785 + 0.0000i   0.4218 - 0.0000i   0.7431 - 0.0000i
   0.5469 - 0.0000i   0.9157 - 0.0000i   0.3922 - 0.0000i
   0.9575 + 0.0000i   0.7922 + 0.0000i   0.6555 + 0.0000i
   0.9649 + 0.0000i   0.9595 + 0.0000i   0.1712 - 0.0000i
```

Algorithms

If the `Wavefront` property value is `'Plane'`, `phased.WidebandCollector` does the following for each plane wave signal:

1. Decomposes the signal into multiple subbands.
2. Uses the phase approximation of the time delays across collecting elements in the far field for each subband.
3. Regroups the collected signals in all the subbands to form the output signal.

If the `Wavefront` property value is `'Unspecified'`, the object collects each channel independently.

For further details, see [1].

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

phased.WidebandFreeSpace

Package: `phased`

Wideband free-space propagation

Description

The System object models wideband signal propagation from one point to another in a free-space environment. The System object applies range-dependent time delay, gain adjustment, and phase shift to the input signal. The object accounts for Doppler shift when either the source or destination is moving. A free-space environment is a boundary-free medium with a speed of signal propagation independent of position and direction. The signal propagates along a straight line from source to destination. For example, you can use this object to model the two-way propagation of a signal from a radar to a target.

For nonpolarized signals, the System object lets you propagate signals from a single point to multiple points or from multiple points to a single point. Multiple-point–to–multiple-point propagation is not supported.

To compute the propagated signal in free space:

1. Define and set up your wideband free space environment as shown in the “Construction” on page 1-2009 section.
2. Call `step` to propagate the signal through free space according to the properties of the System object. The behavior of `step` is specific to each object in the toolbox.

When propagating a round trip signal in free space, you can use one `WidebandFreeSpace` System object to compute the two-way propagation delay. Alternatively, you can use two separate `WidebandFreeSpace` System objects to compute one-way propagation delays in each direction. Due to filter distortion, the total round trip delay when you employ two-way propagation can differ from the delay when you use two one-way `phased.WidebandFreeSpace` System objects. It is more accurate to use a single two-way `phased.WidebandFreeSpace` System object. To set this option, use the `TwoWayPropagation` property.

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

Construction

`sWBFS = phased.WidebandFreeSpace` creates a wideband free space System object, `sWBFS`.

`sWBFS = phased.WidebandFreeSpace(Name,Value)` creates a wideband free space System object, `sWBFS`, with each specified property `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as `Name1`,`Value1`,...,`NameN`,`ValueN`.

Properties

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `double`

`TwoWayPropagation` — Enable two-way propagation
`false` (default) | `true`

Enable two-way propagation, specified as a `false` or `true`. Set this property to `true` to perform round-trip propagation between the signal origin and destination specified in `step`. Set this property to `false` to perform only one-way propagation from the origin to the destination.

Example: `true`

Data Types: `logical`

`SampleRate` — Sample rate of signal
`1e6` (default) | positive scalar

Sample rate of signal, specified as a positive scalar. Units are in Hz. The System object uses this quantity to calculate the propagation delay in units of samples.

Example: `1e6`

Data Types: `double`

`NumSubbands` — Number of processing subbands
`64` (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: `128`

Data Types: `double`

`MaximumDistanceSource` — Source of maximum one-way propagation distance
`'Auto'` (default) | `'Property'`

Source of maximum one-way propagation distance, specified as `'Auto'` or `'Property'`. The maximum one-way propagation distance is used to allocate sufficient memory for signal delay computation. When you set this property to `'Auto'`, the System object automatically allocates memory. When you set this property to 'Property', you specify the maximum one-way propagation distance using the value of the MaximumDistance property.

Data Types: char

MaximumDistance — Maximum one-way propagation distance
10000 (default) | positive real-valued scalar

Maximum one-way propagation distance, specified as a positive real-valued scalar. Units are in meters. Any signal that propagates more than the maximum one-way distance is ignored. The maximum distance must be greater than or equal to the largest position-to-position distance.

Example: 5000

Dependencies

To enable this property, set the MaximumDistanceSource property to 'Property'.

Data Types: double

MaximumNumInputSamplesSource — Source of maximum number of samples
'Auto' (default) | 'Property'

The source of the maximum number of samples of the input signal, specified as 'Auto' or 'Property'. When you set this property to 'Auto', the propagation model automatically allocates enough memory to buffer the input signal. When you set this property to 'Property', you specify the maximum number of samples in the input signal using the MaximumNumInputSamples property. Any input signal longer than that value is truncated.

To use this object with variable-size signals in a MATLAB Function Block in Simulink, set the MaximumNumInputSamplesSource property to 'Property' and set a value for the MaximumNumInputSamples property.

Example: 'Property'

Dependencies

To enable this property, set MaximumDistanceSource to 'Property'.

Data Types: char

MaximumNumInputSamples — Maximum number of input signal samples
100 (default) | positive integer

Maximum number of input signal samples, specified as a positive integer. The input signal is the first argument of the step method, after the System object itself. The size of the input signal is the number of rows in the input matrix. Any input signal longer than this number is truncated. To process signals completely, ensure that this property value is greater than any maximum input signal length.

The waveform-generating System objects determine the maximum signal size:

- For any waveform, if the waveform OutputFormat property is set to 'Samples', the maximum signal length is the value specified in the NumSamples property.
- For pulse waveforms, if the OutputFormat is set to 'Pulses', the signal length is the product of the smallest pulse repetition frequency, the number of pulses, and the sample rate.
- For continuous waveforms, if the OutputFormat is set to 'Sweeps', the signal length is the product of the sweep time, the number of sweeps, and the sample rate.

Example: 2048

Dependencies

To enable this property, set `MaximumNumInputSamplesSource` to `'Property'`.

Data Types: `double`

Methods

reset	Reset states of phased.WidebandFreeSpace System object
step	Propagate wideband signal from point to point using free-space channel model

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Free-Space Propagation of Wideband Signals

Propagate a wideband signal with three tones in an underwater acoustic with constant speed of propagation. You can model this environment as free space. The center frequency is 100 kHz and the frequencies of the three tones are 75 kHz, 100 kHz, and 125 kHz, respectively. Plot the spectrum of the original signal and the propagated signal to observe the Doppler effect. The sampling frequency is 100 kHz.

```
c = 1500;
fc = 100e3;
fs = 100e3;
relfreqs = [-25000,0,25000];
```

Set up a stationary radar and moving target and compute the expected Doppler.

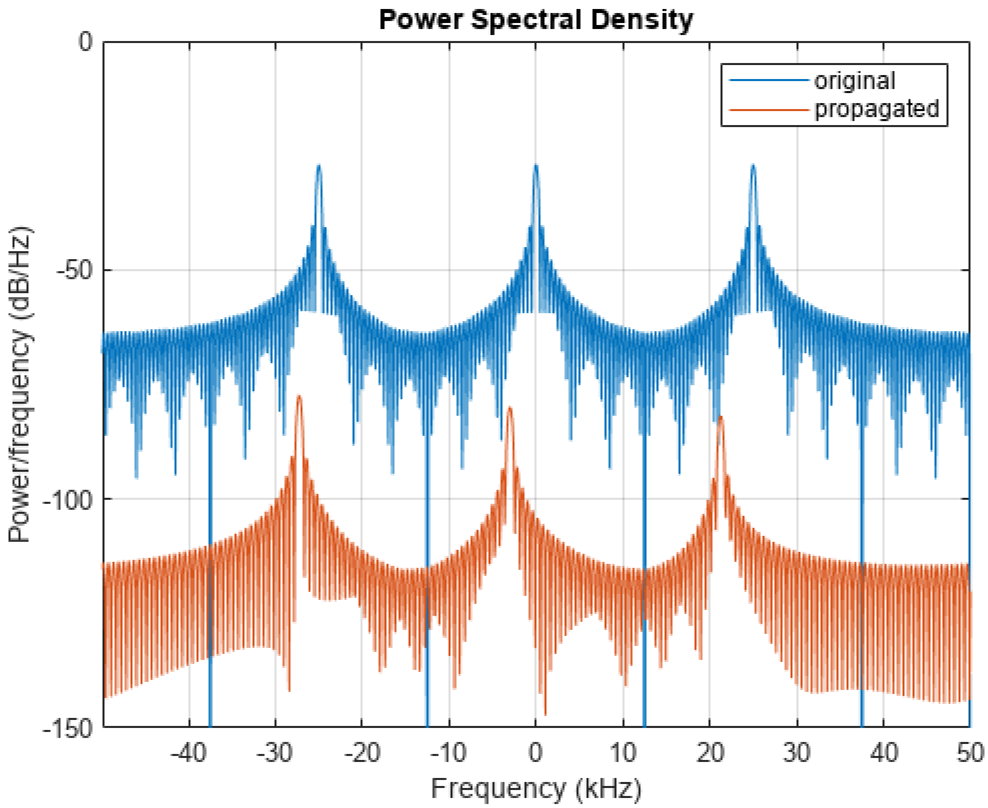
```
rpos = [0;0;0];
rvel = [0;0;0];
tpos = [30/fs*c; 0;0];
tvel = [45;0;0];
dop = -tvel(1)./(c./(relfreqs + fc));
```

Create a signal and propagate the signal to the moving target.

```
t = (0:199)/fs;
x = sum(exp(1i*2*pi*t.'*relfreqs),2);
channel = phased.WidebandFreeSpace(...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc,...
    'SampleRate',fs);
y = channel(x,rpos,tpos,rvel,tvel);
```

Plot the spectra of the original signal and the Doppler-shifted signal.

```
periodogram([x y],rectwin(size(x,1)),1024,fs,'centered')
ylim([-150 0])
legend('original','propagated');
```



For this wideband signal, you can see that the magnitude of the Doppler shift increases with frequency. In contrast, for narrowband signals, the Doppler shift is assumed constant over the band.

More About

Free-space Time Delay and Path Loss

When the origin and destination are stationary relative to each other, you can write the output signal of a free-space channel as $Y(t) = x(t\text{-}\tau)/L_{fsp}$. The quantity τ is the signal delay and L_{fsp} is the free-space path loss. The delay τ is given by R/c, where R is the propagation distance and c is the propagation speed. The free-space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far field of the transmitting element or array. In the near field, the free-space path loss formula is not valid and can result in a loss smaller than one, equivalent to a signal gain. Therefore, the loss is set to unity for range values, $R \leq \lambda/4\pi$.

When the origin and destination have relative motion, the processing also introduces a Doppler frequency shift. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The quantity v is the relative speed of the destination with respect to the origin.

For more details on free space channel propagation, see [2].

Subband Frequency Processing

Subband processing decomposes a wideband signal into multiple subbands and applies narrowband processing to the signal in each subband. The signals for all subbands are summed to form the output signal.

When using wideband frequency System objects or blocks, you specify the number of subbands, N_B, in which to decompose the wideband signal. Subband center frequencies and widths are automatically computed from the total bandwidth and number of subbands. The total frequency band is centered on the carrier or operating frequency, f_c. The overall bandwidth is given by the sample rate, f_s. Frequency subband widths are $\Delta f = f_s/N_B$. The center frequencies of the subbands are

$$f_m = \begin{cases} f_c - \dfrac{f_s}{2} + (m-1)\Delta f, & N_B \text{ even} \\ f_c - \dfrac{(N_B-1)f_s}{2N_B} + (m-1)\Delta f, & N_B \text{ odd} \end{cases}, \quad m = 1, \dots, N_B$$

Some System objects let you obtain the subband center frequencies as output when you run the object. The returned subband frequencies are ordered consistently with the ordering of the discrete Fourier transform. Frequencies above the carrier appear first, followed by frequencies below the carrier.

The `phased.WidebandFreeSpace` System object uses narrowband time delay and loss algorithms for each subband.

Version History

Introduced in R2015b

References

[1] Proakis, J. *Digital Communications*. New York: McGraw-Hill, 2001.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`twoRayChannel` | `phased.FreeSpace` | `phased.WidebandRadiator` | `phased.WidebandCollector` | `phased.RadarTarget` | `fspl`

reset

System object: `phased.WidebandFreeSpace`
Package: `phased`

Reset states of phased.WidebandFreeSpace System object

Syntax

`reset(sWBFS)`

Description

`reset(sWBFS)` resets the internal state of the `phased.WidebandFreeSpace` object, `sWBFS`. If the `SeedSource` property applies and has the value `'Property'`, then this method resets the random number generator state.

Input Arguments

`sWBFS` — Wideband free space propagator
System object

Wideband free space propagator, specified as a System object.

Example: `phased.WidebandFreeSpace`

Version History

Introduced in R2015b

step

System object: `phased.WidebandFreeSpace`
Package: `phased`

Propagate wideband signal from point to point using free-space channel model

Syntax

```
prop_sig = step(sWBFS,sig,origin_pos,dest_pos,origin_vel,dest_vel)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`prop_sig = step(sWBFS,sig,origin_pos,dest_pos,origin_vel,dest_vel)` returns the resulting signal, `prop_sig`, when a wideband signal `sig` propagates through a free-space channel from the `origin_pos` position to the `dest_pos` position. Either the `origin_pos` or `dest_pos` arguments can specify more than one point but you cannot specify both as having multiple points. The velocity of the signal origin is specified in `origin_vel` and the velocity of the signal destination is specified in `dest_vel`. The dimensions of `origin_vel` and `dest_vel` must agree with the dimensions of `origin_pos` and `dest_pos`, respectively.

Electromagnetic fields propagated through a free-space channel can be polarized or nonpolarized. For nonpolarized fields, such as acoustic fields, the propagating signal field, `sig`, is a vector or matrix. When the fields are polarized, `sig` is a `struct` array. Every structure element represents an electric field vector signal.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

`sWBFS` — Wideband free space propagator

System object

Wideband free space propagator, specified as a System object.

Example: `phased.WidebandFreeSpace`

`sig` — Wideband signal

M-by-*N* complex-valued matrix | *1*-by-*N* `struct` array containing complex-valued fields

- Wideband nonpolarized signal, specified as an *M*-by-*N* complex-valued matrix. Each column contains a signal propagated along one of the free-space paths.
- Wideband polarized signal, specified as a 1-by-*N* `struct` array containing complex-valued fields. Each `struct` element contains an *M*-by-1 column vector of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing a polarized signal propagating along one of the free-space paths.

The quantity *M* is the number of signal samples and *N* is the number of free-space channels. Each channel corresponds to a source-destination pair.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

For polarized fields, each `struct` element contains three *M*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Data Types: `double`
Complex Number Support: Yes

`origin_pos` — Signal origin
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Origin of the signal or signals, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. Position units are in meters. The quantity *N* is the number of free-space channels. If `origin_pos` is a column vector, it takes the form `[x;y;z]`. If `origin_pos` is a matrix, each column specifies a different signal origin and has the form `[x;y;z]`.

You cannot specify both `origin_pos` and `dest_pos` as matrices. At least one must be a 3-by-1 column vector.

Example: `[1000;100;500]`

Data Types: `double`

`dest_pos` — Signal destination
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Destination of the signal or signals, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. Position units are in meters. The quantity *N* is the number of free-space channels. If `dest_pos` is a 3-by-1 column vector, it takes the form `[x;y;z]`. If `dest_pos` is a matrix, each column specifies a different signal destination and takes the form `[x;y;z]`.

You cannot specify both `origin_pos` and `dest_pos` as matrices. At least one must be a 3-by-1 column vector.

Example: `[0;0;0]`

Data Types: `double`

origin_vel — Signal origin velocity
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Velocity of signal origin, specified as a real-valued 3-by-1 column vector or real-valued 3-by-*N* matrix. Velocity units are in meters per second. The dimension of `origin_vel` must match the dimension of `origin_pos`. If `origin_vel` is a column vector, it takes the form `[Vx;Vy;Vz]`. If `origin_vel` is a 3–by-*N* matrix, each column specifies a different origin velocity and has the form `[Vx;Vy;Vz]`.

Example: `[10;0;5]`

Data Types: `double`

dest_vel — Signal destination velocity
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Velocity of signal destinations, specified as a 3-by-1 column vector or 3–by-*N* matrix. Velocity units are in meters per second. The dimension of `dest_vel` must match the dimension of `dest_pos`. If `dest_vel` is a column vector, it takes the form `[Vx;Vy;Vz]`. If `dest_vel` is a 3–by-*N* matrix, each column specifies a different destination velocity and has the form `[Vx;Vy;Vz]`.

Example: `[0;0;0]`

Data Types: `double`

Output Arguments

prop_sig — Wideband propagated signal
M-by-*N* complex-valued matrix | 1-by-*N* `struct` array containing complex-valued fields

- Wideband nonpolarized signal, specified as an *M*-by-*N* complex-valued matrix. Each column contains a signal propagated along one of the free-space paths.
- Wideband polarized signal, specified as a 1-by-*N* `struct` array containing complex-valued fields. Each `struct` element contains an *M*-by-1 column vector of electromagnetic field components `(sig.X,sig.Y,sig.Z)` representing a polarized signal propagating along one of the free-space paths.

The output `prop_sig` contains signal samples arriving at the signal destination within the current `step`time frame. Whenever it takes longer than the current time frame for the signal to propagate from the origin to the destination, the output may not contain all contribution from the input. The next call to `step` will return more of the propagated signal.

Examples

Free-Space Propagation of Wideband Signals

Propagate a wideband signal with three tones in an underwater acoustic with constant speed of propagation. You can model this environment as free space. The center frequency is 100 kHz and the frequencies of the three tones are 75 kHz, 100 kHz, and 125 kHz, respectively. Plot the spectrum of the original signal and the propagated signal to observe the Doppler effect. The sampling frequency is 100 kHz.

```
c = 1500;
fc = 100e3;
```

```
fs = 100e3;
relfreqs = [-25000,0,25000];
```

Set up a stationary radar and moving target and compute the expected Doppler.

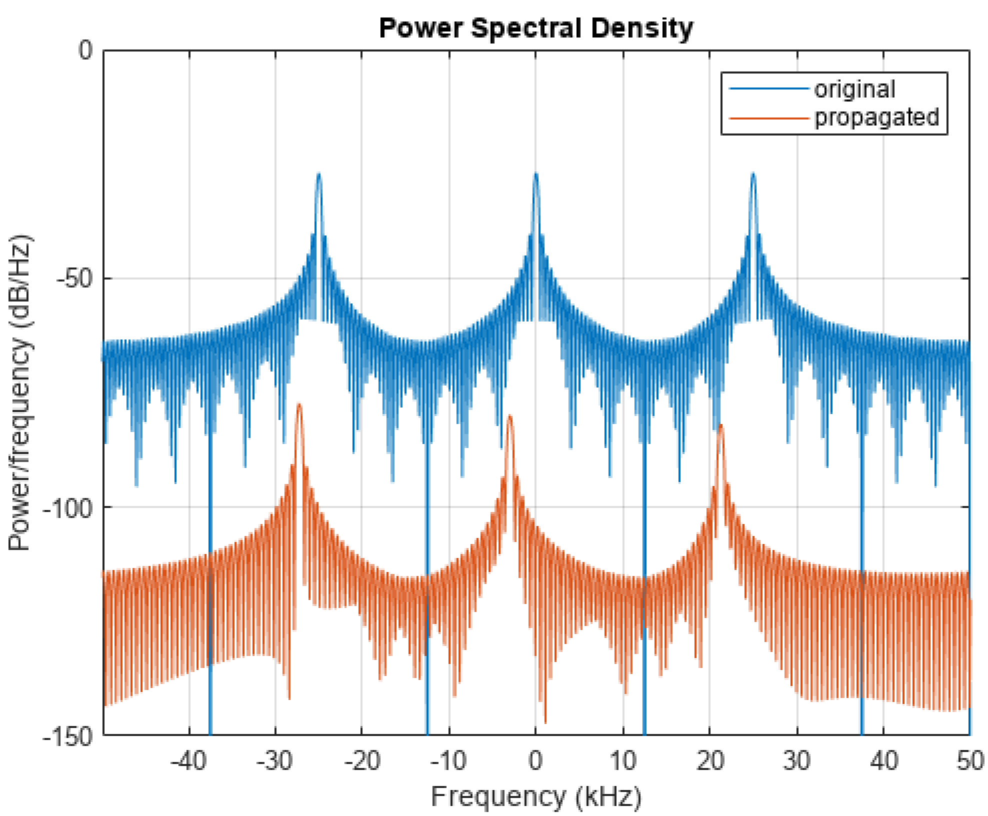
```
rpos = [0;0;0];
rvel = [0;0;0];
tpos = [30/fs*c; 0;0];
tvel = [45;0;0];
dop = -tvel(1)./(c./(relfreqs + fc));
```

Create a signal and propagate the signal to the moving target.

```
t = (0:199)/fs;
x = sum(exp(1i*2*pi*t.'*relfreqs),2);
channel = phased.WidebandFreeSpace(...
    'PropagationSpeed',c,...
    'OperatingFrequency',fc,...
    'SampleRate',fs);
y = channel(x,rpos,tpos,rvel,tvel);
```

Plot the spectra of the original signal and the Doppler-shifted signal.

```
periodogram([x y],rectwin(size(x,1)),1024,fs,'centered')
ylim([-150 0])
legend('original','propagated');
```



For this wideband signal, you can see that the magnitude of the Doppler shift increases with frequency. In contrast, for narrowband signals, the Doppler shift is assumed constant over the band.

Version History

Introduced in R2015b

References

[1] Proakis, J. *Digital Communications*. New York: McGraw-Hill, 2001.

[2] Skolnik, M. *Introduction to Radar Systems*. 3rd Ed. New York: McGraw-Hill

[3] Saakian, A. *Radio Wave Propagation Fundamentals*. Norwood, MA: Artech House, 2011.

[4] Balanis, C. *Advanced Engineering Electromagnetics*. New York: Wiley & Sons, 1989.

[5] Rappaport, T. *Wireless Communications: Principles and Practice*. 2nd Ed. New York: Prentice Hall, 2002.

phased.WidebandLOSChannel

Package: phased

Wideband LOS propagation channel

Description

The phased.WidebandLOSChannel models the propagation of narrowband electromagnetic signals through a line-of-sight (LOS) channel from a source to a destination. In an LOS channel, propagation paths are straight lines from point to point. The propagation model in the LOS channel includes free-space attenuation in addition to attenuation due to atmospheric gases, rain, fog, and clouds. You can use phased.WidebandLOSChannel to model the propagation of signals between multiple points simultaneously. The System object works for all frequencies.

While the attenuation models for atmospheric gases and rain are valid for electromagnetic signals in the frequency range 1–1000 GHz only, the attenuation model for fog and clouds is valid for 10–1000 GHz. Outside these frequency ranges, the System object uses the nearest valid value.

The phased.WidebandLOSChannel System object applies range-dependent time delays to the signals, as well as gains or losses. When either the source or destination is moving, the System object applies Doppler shifts.

Like the phased.WidebandFreeSpace System object, the phased.WidebandLOSChannel System object supports two-way propagation.

To compute the propagation delay for specified source and receiver points:

1. Define and set up your Wideband LOS channel using the "Construction" on page 1-2021 procedure. You can set the System object properties during construction or leave them at their default values.
2. Call the step method to compute the propagated signal using the properties of the phased.WidebandLOSChannel System object. You can change tunable properties before or after any call to the step method.

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

Construction

sWBLOS = phased.WidebandLOSChannel creates a Wideband LOS attenuating propagation channel System object, sWBLOS.

sWBLOS = phased.WidebandLOSChannel(Name,Value) creates a System object, sWBLOS, with each specified property Name set to the specified Value. You can specify additional name and value pair arguments in any order as (Name1,Value1,...,NameN,ValueN).

Properties

`PropagationSpeed` — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`OperatingFrequency` — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `1e9`

Data Types: `double`

`SpecifyAtmosphere` — Enable atmospheric attenuation model
`false` (default) | `true`

Option to enable the atmospheric attenuation model, specified as a `false` or `true`. Set this property to `true` to add signal attenuation caused by atmospheric gases, rain, fog, or clouds. Set this property to `false` to ignore atmospheric effects in propagation.

Setting `SpecifyAtmosphere` to `true`, enables the `Temperature`, `DryAirPressure`, `WaterVapourDensity`, `LiquidWaterDensity`, and `RainRate` properties.

Data Types: `logical`

`Temperature` — Ambient temperature
`15` (default) | real-valued scalar

Ambient temperature, specified as a real-valued scalar. Units are in degrees Celsius.

Example: `20.0`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

`DryAirPressure` — Atmospheric dry air pressure
`101.325e3` (default) | positive real-valued scalar

Atmospheric dry air pressure, specified as a positive real-valued scalar. Units are in pascals (Pa). The default value of this property corresponds to one standard atmosphere.

Example: `101.0e3`

Dependencies

To enable this property, set `SpecifyAtmosphere` to `true`.

Data Types: `double`

WaterVapourDensity — Atmospheric water vapor density
7.5 (default) | positive real-valued scalar

Atmospheric water vapor density, specified as a positive real-valued scalar. Units are in g/m^3.

Example: 7.4

Dependencies

To enable this property, set SpecifyAtmosphere to true.

Data Types: double

LiquidWaterDensity — Liquid water density
0.0 (default) | nonnegative real-valued scalar

Liquid water density of fog or clouds, specified as a nonnegative real-valued scalar. Units are in g/m^3. Typical values for liquid water density are 0.05 for medium fog and 0.5 for thick fog.

Example: 0.1

Dependencies

To enable this property, set SpecifyAtmosphere to true.

Data Types: double

RainRate — Rainfall rate
0.0 (default) | non-negative real-valued scalar

Rainfall rate, specified as a nonnegative real-valued scalar. Units are in mm/hr. This property applies only when you set SpecifyAtmosphere to true.

Example: 10.0

Data Types: double

TwoWayPropagation — Enable two-way propagation
false (default) | true

Enable two-way propagation, specified as a false or true. Set this property to true to perform round-trip propagation between the signal origin and destination specified in step. Set this property to false to perform only one-way propagation from the origin to the destination.

Example: true

Data Types: logical

SampleRate — Sample rate of signal
1e6 (default) | positive scalar

Sample rate of signal, specified as a positive scalar. Units are in Hz. The System object uses this quantity to calculate the propagation delay in units of samples.

Example: 1e6

Data Types: double

NumSubbands — Number of processing subbands
64 (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: 128

Data Types: double

MaximumDistanceSource — Source of maximum one-way propagation distance
'Auto' (default) | 'Property'

Source of maximum one-way propagation distance, specified as 'Auto' or 'Property'. The maximum one-way propagation distance is used to allocate sufficient memory for signal delay computation. When you set this property to 'Auto', the System object automatically allocates memory. When you set this property to 'Property', you specify the maximum one-way propagation distance using the value of the MaximumDistance property.

Data Types: char

MaximumDistance — Maximum one-way propagation distance
10000 (default) | positive real-valued scalar

Maximum one-way propagation distance, specified as a positive real-valued scalar. Units are in meters. Any signal that propagates more than the maximum one-way distance is ignored. The maximum distance must be greater than or equal to the largest position-to-position distance.

Example: 5000

Dependencies

To enable this property, set the MaximumDistanceSource property to 'Property'.

Data Types: double

MaximumNumInputSamplesSource — Source of maximum number of samples
'Auto' (default) | 'Property'

The source of the maximum number of samples of the input signal, specified as 'Auto' or 'Property'. When you set this property to 'Auto', the propagation model automatically allocates enough memory to buffer the input signal. When you set this property to 'Property', you specify the maximum number of samples in the input signal using the MaximumNumInputSamples property. Any input signal longer than that value is truncated.

To use this object with variable-size signals in a MATLAB Function Block in Simulink, set the MaximumNumInputSamplesSource property to 'Property' and set a value for the MaximumNumInputSamples property.

Example: 'Property'

Dependencies

To enable this property, set MaximumDistanceSource to 'Property'.

Data Types: char

MaximumNumInputSamples — Maximum number of input signal samples
100 (default) | positive integer

Maximum number of input signal samples, specified as a positive integer. The input signal is the first argument of the step method, after the System object itself. The size of the input signal is the number of rows in the input matrix. Any input signal longer than this number is truncated. To process signals completely, ensure that this property value is greater than any maximum input signal length.

The waveform-generating System objects determine the maximum signal size:

- For any waveform, if the waveform `OutputFormat` property is set to `'Samples'`, the maximum signal length is the value specified in the `NumSamples` property.
- For pulse waveforms, if the `OutputFormat` is set to `'Pulses'`, the signal length is the product of the smallest pulse repetition frequency, the number of pulses, and the sample rate.
- For continuous waveforms, if the `OutputFormat` is set to `'Sweeps'`, the signal length is the product of the sweep time, the number of sweeps, and the sample rate.

Example: `2048`

Dependencies

To enable this property, set `MaximumNumInputSamplesSource` to `'Property'`.

Data Types: `double`

Methods

reset	Reset states of System object
step	Propagate signal in Wideband LOS channel

Common to All System Objects	
`release`	Allow System object property value changes

Examples

Spectrum of Propagated Signal in Wideband LOS Channel

Propagate a wideband signal in a line-of-sight (LOS) channel from a radar at *(0,0,0)* meters to a target at *(35,0,0)* meters in medium fog. Set the fog liquid water density to 0.05 gm/m3. Assume rain is falling at 5 mm/hr. The signal carrier frequency is 20 GHz. The signal is a sum of four cw tones at 19.75, 19.875, 20.125, and 20.25 GHz. Set the signal duration to 0.5 μs and the sample rate to 2.0 GHz. Assume the radar is stationary and the target approaches the radar at 40 m/s. The atmospheric temperature is 12°C.

Set the signal parameters and create the transmitted signal.

```
c = physconst('LightSpeed');
fs = 2e9;
freq = [-0.25,-.125,0.125,0.25]*1e9;
fc = 20.0e9;
dt = 1/fs;
t = [0:dt:.5e-6];
sig = sum(exp(1i*2*pi*t.'*freq),2);
```

Specify the atmosphere parameters and create the `phased.WidebandChannel` System object™.

```
lwd = 0.05;
rainrate = 5.0;
```

```
temp = 12.0;
loschannel = phased.WidebandLOSChannel('SampleRate',fs,'PropagationSpeed',c,...
    'SpecifyAtmosphere',true,'OperatingFrequency',fc,'RainRate',rainrate,...
    'LiquidWaterDensity',lwd,'Temperature',temp);
```

Specify the radar and target positions and velocities.

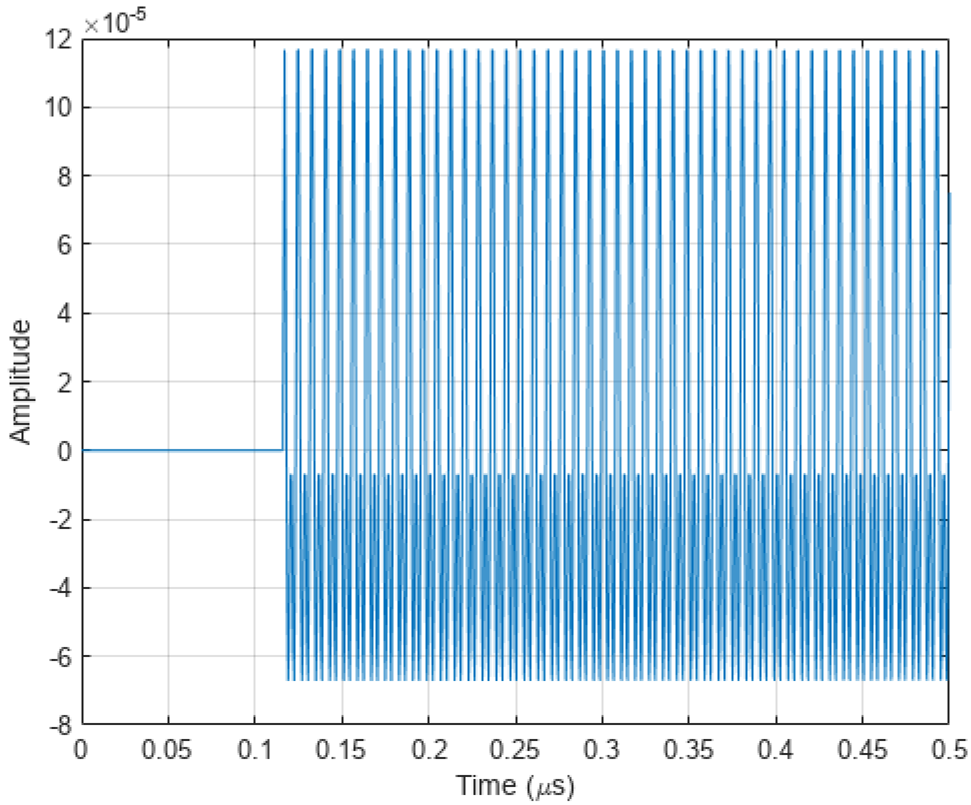
```
xradar = [0,0,0].';
vradar = [0,0,0].';
xtgt = [35,0,0].';
vtgt = [-40,0,0].';
```

Propagate the signal using the `step` method.

```
prop_sig = loschannel(sig,xradar,xtgt,vradar,vtgt);
```

Plot the propagated signal. For a target range of 35 m, the propagation delay is 0.11 µs as seen in the plot.

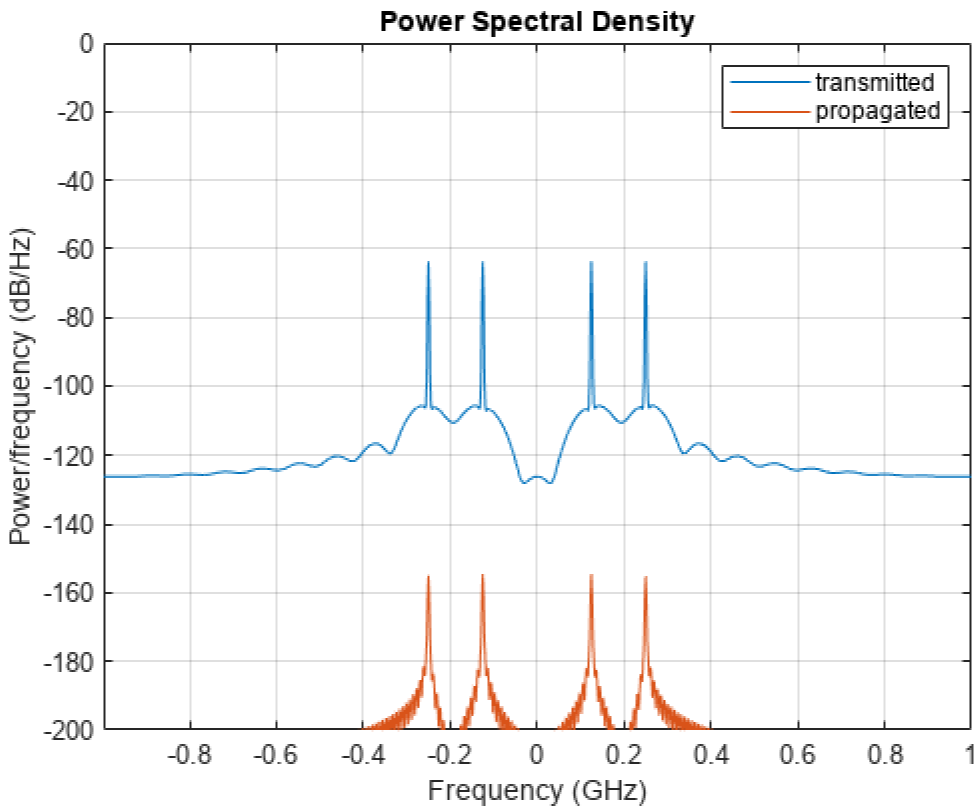
```
plot(t*1e6,real(prop_sig))
grid
xlabel('Time ({\mu}s)')
ylabel('Amplitude')
```



Using the `periodogram` function with a Taylor window, plot the spectra of the original and propagated signals.

```
nfft = 1024;
nsamp = size(sig,1);
```

```
periodogram([sig prop_sig],taylorwin(nsamp),nfft,fs,'centered')
ylim([-200 0])
legend('transmitted','propagated')
```



More About

Attenuation and Loss Factors

Attenuation or path loss in the Wideband LOS channel consists of four components. $L = L_{fsp}L_gL_cL_r$, where

- L_{fsp} is the free-space path attenuation
- L_g is the atmospheric path attenuation
- L_c is the fog and cloud path attenuation
- L_r is the rain path attenuation

Each component is in magnitude units, not in dB.

Free-space Time Delay and Loss

When the origin and destination are stationary relative to each other, you can write the output signal of a free-space channel as $Y(t) = x(t-\tau)/L_{fsp}$. The quantity τ is the signal delay and L_{fsp} is the free-space path loss. The delay τ is given by R/c, where R is the propagation distance and c is the propagation speed. The free-space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far field of the transmitting element or array. In the near field, the free-space path loss formula is not valid and can result in a loss smaller than one, equivalent to a signal gain. Therefore, the loss is set to unity for range values, $R \leq \lambda/4\pi$.

When the origin and destination have relative motion, the processing also introduces a Doppler frequency shift. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The quantity v is the relative speed of the destination with respect to the origin.

For more details on free space channel propagation, see [5].

Atmospheric Gas Attenuation Model

This model calculates the attenuation of signals that propagate through atmospheric gases.

Electromagnetic signals attenuate when they propagate through the atmosphere. This effect is due primarily to the absorption resonance lines of oxygen and water vapor, with smaller contributions coming from nitrogen gas. The model also includes a continuous absorption spectrum below 10 GHz. The ITU model *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* is used. The model computes the specific attenuation (attenuation per kilometer) as a function of temperature, pressure, water vapor density, and signal frequency. The atmospheric gas model is valid for frequencies from 1–1000 GHz and applies to polarized and nonpolarized fields.

The formula for specific attenuation at each frequency is

$$\gamma = \gamma_o(f) + \gamma_w(f) = 0.1820 f N''(f) \, .$$

The quantity $N''()$ is the imaginary part of the complex atmospheric refractivity and consists of a spectral line component and a continuous component:

$$N''(f) = \sum_i S_i F_i + N''_D(f)$$

The spectral component consists of a sum of discrete spectrum terms composed of a localized frequency bandwidth function, $F(f)_i$, multiplied by a spectral line strength, S_i. For atmospheric oxygen, each spectral line strength is

$$S_i = a_1 \times 10^{-7} \left(\frac{300}{T}\right)^3 \exp\left[a_2 (1 - \left(\frac{300}{T}\right)\right] P \, .$$

For atmospheric water vapor, each spectral line strength is

$$S_i = b_1 \times 10^{-1} \left(\frac{300}{T}\right)^{3.5} \exp\left[b_2 (1 - \left(\frac{300}{T}\right)\right] W \, .$$

P is the dry air pressure, W is the water vapor partial pressure, and T is the ambient temperature. Pressure units are in hectoPascals (hPa) and temperature is in degrees Kelvin. The water vapor partial pressure, W, is related to the water vapor density, ρ, by

$$W = \frac{\rho T}{216.7} \, .$$

The total atmospheric pressure is $P + W$.

For each oxygen line, S_i depends on two parameters, a_1 and a_2. Similarly, each water vapor line depends on two parameters, b_1 and b_2. The ITU documentation cited at the end of this section contains tabulations of these parameters as functions of frequency.

The localized frequency bandwidth functions $F_i(f)$ are complicated functions of frequency described in the ITU references cited below. The functions depend on empirical model parameters that are also tabulated in the reference.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length, R. Then, the total attenuation is $L_g= R(\gamma_o + \gamma_w)$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Fog and Cloud Attenuation Model

This model calculates the attenuation of signals that propagate through fog or clouds.

Fog and cloud attenuation are the same atmospheric phenomenon. The ITU model, *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog* is used. The model computes the specific attenuation (attenuation per kilometer), of a signal as a function of liquid water density, signal frequency, and temperature. The model applies to polarized and nonpolarized fields. The formula for specific attenuation at each frequency is

$$\gamma_c = K_l(f)M,$$

where M is the liquid water density in gm/m^3. The quantity $K_l(f)$ is the specific attenuation coefficient and depends on frequency. The cloud and fog attenuation model is valid for frequencies 10–1000 GHz. Units for the specific attenuation coefficient are (dB/km)/(g/m^3).

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length R. Total attenuation is $L_c = R\gamma_c$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply narrowband attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Rainfall Attenuation Model

This model calculates the attenuation of signals that propagate through regions of rainfall. Rain attenuation is a dominant fading mechanism and can vary from location-to-location and from year-to-year.

Electromagnetic signals are attenuated when propagating through a region of rainfall. Rainfall attenuation is computed according to the ITU rainfall model *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. The model computes the specific attenuation (attenuation per kilometer) of a signal as a function of rainfall rate, signal frequency, polarization, and path elevation angle. The specific attenuation, γ_R, is modeled as a power law with respect to rain rate

$$\gamma_R = kR^\alpha,$$

where R is rain rate. Units are in mm/hr. The parameter k and exponent α depend on the frequency, the polarization state, and the elevation angle of the signal path. The specific attenuation model is valid for frequencies from 1–1000 GHz.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the an effective propagation distance, d_{eff}. Then, the total attenuation is $L = d_{\mathrm{eff}}\gamma_{\mathrm{R}}$.

The effective distance is the geometric distance, d, multiplied by a scale factor

$$r = \frac{1}{0.477d^{0.633}R_{0.01}^{0.073\alpha}f^{0.123} - 10.579(1 - \exp(-0.024d))}$$

where f is the frequency. The article *Recommendation ITU-R P.530-17 (12/2017): Propagation data and prediction methods required for the design of terrestrial line-of-sight systems* presents a complete discussion for computing attenuation.

The rain rate, R, used in these computations is the long-term statistical rain rate, $R_{0.01}$. This is the rain rate that is exceeded 0.01% of the time. The calculation of the statistical rain rate is discussed in *Recommendation ITU-R P.837-7 (06/2017): Characteristics of precipitation for propagation modelling*. This article also explains how to compute the attenuation for other percentages from the 0.01% value.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Subband Frequency Processing

Subband processing decomposes a wideband signal into multiple subbands and applies narrowband processing to the signal in each subband. The signals for all subbands are summed to form the output signal.

When using wideband frequency System objects or blocks, you specify the number of subbands, N_{B}, in which to decompose the wideband signal. Subband center frequencies and widths are automatically computed from the total bandwidth and number of subbands. The total frequency band is centered on the carrier or operating frequency, f_c. The overall bandwidth is given by the sample rate, f_s. Frequency subband widths are $\Delta f = f_s/N_{\mathrm{B}}$. The center frequencies of the subbands are

$$f_m = \begin{cases} f_c - \dfrac{f_s}{2} + (m-1)\Delta f, & N_B \text{ even} \\ f_c - \dfrac{(N_B - 1)f_s}{2N_B} + (m-1)\Delta f, & N_B \text{ odd} \end{cases}, \quad m = 1, \ldots, N_B$$

Some System objects let you obtain the subband center frequencies as output when you run the object. The returned subband frequencies are ordered consistently with the ordering of the discrete Fourier transform. Frequencies above the carrier appear first, followed by frequencies below the carrier.

The `phased.WidebandLOSChannel` System object uses narrowband time delay and attenuation algorithms for each subband.

Version History

Introduced in R2016a

References

[1] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases*. 2013.

[2] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog*. 2013.

[3] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. 2005.

[4] Seybold, J. *Introduction to RF Propagation*. New York: Wiley & Sons, 2005.

[5] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

Functions

`rangeangle` | `fogpl` | `gaspl` | `rainpl` | `fspl`

Objects

`phased.FreeSpace` | `phased.RadarTarget` | `phased.BackscatterRadarTarget` | `twoRayChannel` | `phased.WidebandFreeSpace` | `phased.LOSChannel`

reset

System object: `phased.WidebandLOSChannel`
Package: `phased`

Reset states of System object

Syntax

`reset(sWBLOS)`

Description

`reset(sWBLOS)` resets the internal state of the `phased.WidebandLOSChannel` System object, `sWBLOS`. If `SeedSource` is a property of this System object and has the value `'Property'`, then this method resets the random number generator state.

Input Arguments

`sWBLOS` — Wideband LOS channel
`phased.WidebandLOSChannel` System object

Wideband LOS channel, specified as a `phased.WidebandLOSChannel` System object.

Example: `phased.WidebandLOSChannel`

Version History

Introduced in R2016a

step

System object: phased.WidebandLOSChannel
Package: phased

Propagate signal in Wideband LOS channel

Syntax

```
prop_sig = step(sLOS,sig,origin_pos,dest_pos,origin_vel,dest_vel)
```

Description

Note Starting in R2016b, instead of using the step method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, y = step(obj,x) and y = obj(x) perform equivalent operations.

prop_sig = step(sLOS,sig,origin_pos,dest_pos,origin_vel,dest_vel) returns the resulting signal, prop_sig, when a wideband signal, sig, propagates through a line-of-sight (LOS) channel from a source located at the origin_pos position to a destination at the dest_pos position. Only one of the origin_pos or dest_pos arguments can specify multiple positions. The other must contain a single position. The velocity of the signal origin is specified in origin_vel and the velocity of the signal destination is specified in dest_vel. The dimensions of origin_vel and dest_vel must match the dimensions of origin_pos and dest_pos, respectively.

Electromagnetic fields propagating through an LOS channel can be polarized or nonpolarized. For nonpolarized fields, the propagating signal field, sig, is a vector or matrix. For polarized fields, sig is an array of structures. The structure elements represent an electric field vector in Cartesian form.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the release method to unlock the object.

Input Arguments

sWBLOS — Wideband LOS channel

phased.WidebandLOSChannel System object

Wideband LOS channel, specified as a phased.WidebandLOSChannel System object.

Example: phased.WidebandLOSChannel

sig — Wideband signal

M-by-*N* complex-valued matrix | 1-by-*N* struct array containing complex-valued fields

Wideband signal, specified as a matrix or struct array, depending on whether is signal or polarized or nonpolarized. The quantity *M* is the number of samples in the signal, and *N* is the number of wideband LOS channels. Each channel corresponds to a source-destination pair.

- Wideband nonpolarized scalar signal. Specify sig as an *M*-by-*N* complex-valued matrix. Each column contains one signal propagated along the line-of-sight path.
- Wideband polarized signal. Specify sig as a 1-by-*N* struct array containing complex-valued fields. Each struct represents a polarized signal propagated along the line-of-sight path. Each struct element contains three *M*-by-1 complex-valued column vectors, sig.X, sig.Y, and sig.Z. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

Example: [1,1;j,1;0.5,0]

Data Types: double
Complex Number Support: Yes

origin_pos — Signal origins
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Origin of signals, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of LOS channels. If origin_pos is a column vector, it takes the form [x;y;z]. If origin_pos is a matrix, each column specifies a different signal origin and has the form [x;y;z]. Units are in meters.

You cannot specify both origin_pos and dest_pos as matrices. At least one must be a 3-by-1 column vector.

Example: [1000;100;500]

Data Types: double

dest_pos — Signal destinations
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Destination position of the signal or signals, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of LOS channels propagating from or to *N* signal origins. If dest_pos is a 3-by-1 column vector, it takes the form [x;y;z]. If dest_pos is a matrix, each column specifies a different signal destination and takes the form [x;y;z] Position units are in meters.

You cannot specify both origin_pos and dest_pos as matrices. At least one must be a 3-by-1 column vector.

Example: [0;0;0]

Data Types: double

origin_vel — Velocities of signal origins
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Velocity of signal origin, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The dimensions of origin_vel must match the dimensions of origin_pos. If origin_vel is a column vector, it takes the form [Vx;Vy;Vz]. If origin_vel is a 3-by-*N* matrix, each column specifies a different origin velocity and has the form [Vx;Vy;Vz]. Velocity units are in meters per second.

Example: [10;0;5]

Data Types: `double`

`dest_vel` — Velocities of signal destinations
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Velocity of signal destinations, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The dimensions of `dest_vel` must match the dimensions of `dest_pos`. If `dest_vel` is a column vector, it takes the form `[Vx;Vy;Vz]`. If `dest_vel` is a 3-by-*N* matrix, each column specifies a different destination velocity and has the form `[Vx;Vy;Vz]` Velocity units are in meters per second.

Example: `[0;0;0]`

Data Types: `double`

Output Arguments

`prop_sig` — Wideband propagated signal
M-by-*N* complex-valued matrix | 1-by-*N* `struct` array containing complex-valued fields

Wideband signal, returned as a matrix or `struct` array, depending on whether the signal is polarized or nonpolarized. The quantity *M* is the number of samples in the signal and *N* is the number of wideband LOS channels. Each channel corresponds to a source-destination pair.

- Wideband nonpolarized scalar signal. `prop_sig` is an *M*-by-*N* complex-valued matrix.

  The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.
- Wideband polarized scalar signal. `prop_sig` is a 1-by-*N* `struct` array containing complex-valued fields. Each `struct` element contains three *M*-by-1 complex-valued column vectors, `sig.X`, `sig.Y`, and `sig.Z`. These vectors represent the *x*, *y*, and *z* Cartesian components of the polarized signal.

  The size of the first dimension of the matrix fields within the `struct` can vary to simulate a changing signal length such as a pulse waveform with variable pulse repetition frequency.

The `prop_sig` output contains signal samples arriving at the signal destination within the current time frame. The current time frame is the time frame of the input signals to `step`. Whenever it takes longer than the current time frame for the signal to propagate from the origin to the destination, the output might not contain all contributions from the input of the current time frame. The remaining output appears in the next call to `step`.

Examples

Propagate Wideband Signal in LOS Channel

Propagate a wideband signal in a line-of-sight (LOS) channel from a radar at *(0,0,0)* meters to a target at *(60,0,0)* meters in medium fog. Set the fog liquid water density to 0.05 g/m^3. Assume rain is falling at 5 mm/hr. The signal carrier frequency is 20 GHz. The signal is a sum of four cw tones at 19.75, 19.875, 20.125, and 20.25 GHz. Set the signal duration to 0.5 microsecond and the sample rate to 2.0 GHz. Assume the radar is stationary and the target approaches the radar at 40 m/s. The atmospheric temperature is 12°C and the dry air pressure is 101.300 kPa.

Set the signal parameters and create the transmitted signal.

```
c = physconst('LightSpeed');
fs = 2e9;
freq = [-0.25,-.125,0.0,0.125,0.25]*1e9;
fc = 20.0e9;
dt = 1/fs;
t = [0:dt:.5e-6];
sig = sum(exp(1i*2*pi*t.'*freq),2);
```

Specify the atmosphere parameters and create the `phased.WidebandChannel` System object™.

```
lwd = 0.05;
rainrate = 5.0;
dap = 101300.0;
temp = 12.0;
sWBLOS = phased.WidebandLOSChannel('SampleRate',fs,'PropagationSpeed',c,...
    'SpecifyAtmosphere',true,'OperatingFrequency',fc,'RainRate',rainrate,...
    'LiquidWaterDensity',lwd,'Temperature',temp,'DryAirPressure',dap);
```

Specify the radar and target positions and velocities.

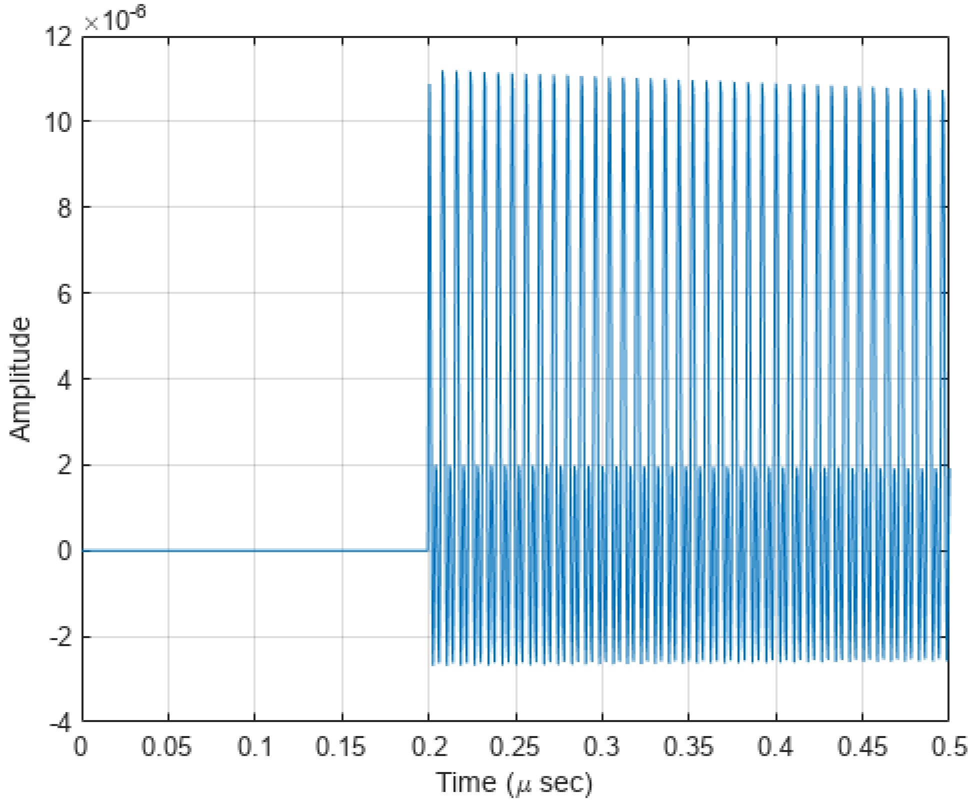
```
xradar = [0,0,0].';
vradar = [0,0,0].';
xtgt = [60,0,0].';
vtgt = [-40,0,0].';
```

Propagate the signal using the `step` method.

```
prop_sig = step(sWBLOS,sig,xradar,xtgt,vradar,vtgt);
```

Plot the propagated signal. For a target range of 60 m, the propagation delay is 0.20 μs as shown in the plot.

```
plot(t*1e6,real(prop_sig))
grid
xlabel('Time (\mu sec)')
ylabel('Amplitude')
```



Version History

Introduced in R2016a

References

[1] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases*. 2013.

[2] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog*. 2013.

[3] Radiocommunication Sector of the International Telecommunication Union. *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. 2005.

[4] Seybold, J. *Introduction to RF Propagation*. New York: Wiley & Sons, 2005.

phased.WidebandRadiator

Package: phased

Wideband signal radiator

Description

The phased.WidebandRadiator System object implements a wideband signal radiator. A radiator converts signals into radiated wavefields transmitted from arrays and individual sensor elements such as antennas, microphone elements, and sonar transducers. The radiator output represents the fields at a reference distance of one meter from the phase center of the element or array. The algorithm divides the signal at each element into frequency subbands and applies a narrowband time-delay to each signal using the phase-shift approximation. Then, the delayed subbands are coherently added to create the output signal. You can then propagate the signals to the far field using, for example, the phased.WidebandFreeSpace or phased.WidebandLOSChannel System objects. You can use this object to

- model radiated signals as polarized or non-polarized fields depending upon whether the element or array supports polarization and the value of the Polarization property. Using polarization, you can transmit a signal as a polarized electromagnetic field, or transmit two independent signals using dual polarizations.
- model acoustic radiated fields by using nonpolarized microphone and sonar transducer array elements and by setting the “Polarization” on page 1-0 to 'None'. You must also set the PropagationSpeed to a value appropriate for the medium.
- radiate fields from subarrays created by the phased.ReplicatedSubarray and phased.PartitionedArray objects. You can steer all subarrays in the same direction using the Steering angle argument, STEERANG, or steer each subarray in a different direction using the Subarray element weights argument, WS. The radiator distributes the signal powers equally among the elements of each subarray.

To radiate signals:

1. Create the phased.WidebandRadiator object and set its properties.
2. Call the object with arguments, as if it were a function.

To learn more about how System objects work, see What Are System Objects?

Creation

Syntax

```
radiator = phased.WidebandRadiator
radiator = phased.WidebandRadiator(Name,Value)
```

Description

radiator = phased.WidebandRadiator creates a wideband signal radiator object, radiator, with default property values.

`radiator = phased.WidebandRadiator(Name,Value)` creates a wideband signal radiator with each property `Name` set to a specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). Enclose each property name in single quotes.

Example: `radiator = phased.WidebandRadiator('Sensor',phased.URA,'CarrierFrequency',300e6)` sets the sensor array to a uniform rectangular array (URA) with default URA property values. The beamformer has a carrier frequency of 300 MHz.

Properties

Unless otherwise indicated, properties are *nontunable*, which means you cannot change their values after calling the object. Objects lock when you call them, and the `release` function unlocks them.

If a property is *tunable*, you can change its value at any time.

For more information on changing property values, see System Design in MATLAB Using System Objects.

`Sensor` — Sensor element or sensor array

`phased.ULA` array with default property values (default) | Phased Array System Toolbox sensor or array

Sensor element or sensor array, specified as a System object belonging to Phased Array System Toolbox. A sensor array can contain subarrays.

Example: `phased.URA`

`PropagationSpeed` — Signal propagation speed

`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second. The default propagation speed is the value returned by `physconst('LightSpeed')`. See `physconst` for more information.

Example: `3e8`

Data Types: `double`

`SampleRate` — Signal sample rate

`1e6` (default) | positive real-valued scalar

Signal sample rate, specified as a positive real-valued scalar. Units are in hertz.

Example: `1e6`

Data Types: `double`

`CarrierFrequency` — Carrier frequency

`1e9` (default) | positive real-valued scalar

Carrier frequency, specified as a positive real-valued scalar. Units are in hertz.

Example: `1e6`

Data Types: `double`

`NumSubbands` — Number of processing subbands
`64` (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: `128`

Data Types: `double`

`SensorGainMeasure` — Specify sensor gain
`'dB'` (default) | `'dBi'`

Sensor gain measure, specified as `'dB'` or `'dBi'`.

- When you set this property to `'dB'`, the input signal power is scaled by the sensor power pattern (in dB) at the corresponding direction and then combined.
- When you set this property to `'dBi'`, the input signal power is scaled by the directivity pattern (in dBi) at the corresponding direction and then combined. This option is useful when you want to compare results with the values computed by the radar equation that uses dBi to specify the antenna gain. The computation using the `'dBi'` option is expensive as it requires an integration over all directions to compute the total radiated power of the sensor.

Data Types: `char`

`Polarization` — Polarization configuration
`'None'` (default) | `'Combined'` | `'Dual'`

Polarization configuration, specified as `'None'`, `'Combined'`, or `'Dual'`. When you set this property to `'None'`, the output field is considered a scalar field. When you set this property to `'Combined'`, the radiated fields are polarized and are interpreted as a single signal in the sensor's inherent polarization. When you set this property to `'Dual'`, the *H* and *V* polarization components of the radiated field are independent signals.

Example: `'Dual'`

Data Types: `char`

`WeightsInputPort` — Enable weights input
`false` (default) | `true`

Enable weights input, specified as `false` or `true`. When `true`, use the object input argument `W` to specify weights. Weights are applied to individual array elements (or at the subarray level when subarrays are supported).

Data Types: `logical`

Usage

Syntax

```
Y = radiator(X,ANG)
Y = radiator(X,ANG,LAXES)
Y = radiator(XH,XV,ANG,LAXES)
Y = radiator( ___ ,W)
Y = radiator( ___ ,STEERANG)
```

```
Y = radiator( ___ ,WS)
Y = radiator(X,ANG,LAXES,W,STEERANG)
```

Description

`Y = radiator(X,ANG)` radiates the signal `X` in the directions specified by `ANG`. For each direction, the method computes the radiated signal, `Y`, by summing the contributions of each element or subarray.

`Y = radiator(X,ANG,LAXES)` also specifies the local coordinate system of the radiator, `LAXES`. This syntax applies when you set the Polarization property to `'Combined'`.

`Y = radiator(XH,XV,ANG,LAXES)` specifies a horizontal-polarization port signal, `XH`, and a vertical-polarization port signal, `XV`. To use this syntax, set the Polarization property to `'Dual'`.

`Y = radiator( ___ ,W)` also specifies `W` as array element or subarray weights. To use this syntax, set the WeightsInputPort property to `true`.

`Y = radiator( ___ ,STEERANG)` also specifies `STEERANG` as the subarray steering angle. To use this syntax, set the Sensor property to an array that supports subarrays and set the `SubarraySteering` property of that array to either `'Phase'` or `'Time'`.

`Y = radiator( ___ ,WS)` also specifies `WS` as the weights applied to each element within each subarray. To use this syntax, set the Sensor property to an array that supports subarrays and set the `SubarraySteering` of that array to `'Custom'`.

You can combine optional input arguments when their enabling properties are set, for example, `Y = radiator(X,ANG,LAXES,W,STEERANG)` combines several input arguments. Optional inputs must be listed in the same order as the order of the enabling properties.

Input Arguments

X — Signal to radiate

complex-valued *M*-by-1 vector | complex-valued *M*-by-*N* matrix

Signal to radiate, specified as a complex-valued *M*-by-1 vector or complex-valued *M*-by-*N* matrix. *M* is the length of the signal, and *N* is the number of array elements (or subarrays when subarrays are supported).

Dimensions of X

Dimension	Signal
M-by-1 vector	The same signal is radiated from all array elements (or all subarrays when subarrays are supported).
M-by-*N* matrix	Each column corresponds to the signal radiated by the corresponding array element (or corresponding subarrays when subarrays are supported).

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this argument, set the Polarization property to `'None'` or `'Combined'`.

Data Types: `double`
Complex Number Support: Yes

`ANG` — Radiating directions of signals
real-valued 2-by-*L* matrix

Radiating directions of signals, specified as a real-valued 2-by-*L* matrix. Each column specifies a radiating direction in the form `[AzimuthAngle;ElevationAngle]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive. Units are in degrees.

Example: `[30,20;45,0]`

Data Types: `double`

`LAXES` — Local coordinate system
real-valued 3-by-3 orthogonal matrix

Local coordinate system, specified as a real-valued 3-by-3 orthogonal matrix. The matrix columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes with respect to the global coordinate system.

Example: `rotx(30)`

Dependencies

To enable this argument, set the `Polarization` property to `'Combined'` or `'Dual'`.

Data Types: `double`

`XH` — H-polarization port signal to radiate
complex-valued *M*-by-1 vector | complex-valued *M*-by-*N* matrix

H-polarization port signal to radiate, specified as a complex-valued *M*-by-1 vector or complex-valued *M*-by-*N* matrix. *M* is the length of the signal, and *N* is the number of array elements (or subarrays when subarrays are supported).

Dimensions of XH

Dimension	Signal
M-by-1 vector	The same signal is radiated from all array elements (or all subarrays when subarrays are supported).
`M-by-N matrix`	Each column corresponds to the signal radiated by the corresponding array element (or corresponding subarrays when subarrays are supported).

The dimensions and sizes of XH and XV must be the same.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this argument, set the Polarization property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

XV — V-polarization port signal to radiate

complex-valued M-by-1 vector | complex-valued M-by-N matrix

V-polarization port signal to radiate, specified as a complex-valued M-by-1 vector or complex-valued M-by-N matrix. M is the length of the signal, and N is the number of array elements (or subarrays when subarrays are supported).

Dimensions of XV

Dimension	Signal
M-by-1 vector	The same signal is radiated from all array elements (or all subarrays when subarrays are supported).
M-by-N matrix	Each column corresponds to the signal radiated by the corresponding array element (or corresponding subarrays when subarrays are supported).

The dimensions and sizes of `XH` and `XV` must be the same.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this argument, set the Polarization property to `'Dual'`.

Data Types: `double`
Complex Number Support: Yes

W — Element or subarray weights

N-by-1 column vector

Element or subarray weights, specified as a complex-valued N-by-1 column vector where N is the number of array elements (or subarrays when the array supports subarrays).

Dependencies

To enable this argument, set the WeightsInputPort property to `true`.

Data Types: `double`
Complex Number Support: Yes

WS — Subarray element weights

complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray element weights

Sensor Array	Subarray weights
`phased.ReplicatedSubarray`	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of `WS` specifies the weights for the corresponding subarray.
`phased.PartitionedArray`	Subarrays may not have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this argument, set the `Sensor` property to an array that contains subarrays and set the `SubarraySteering` property of the array to `'Custom'`.

Data Types: `double`
Complex Number Support: Yes

`STEERANG` — Subarray steering angle

real-valued 2-by-1 vector

Subarray steering angle, specified as a length-2 column vector. The vector has the form `[azimuthAngle;elevationAngle]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive. Units are in degrees.

Example: `[20;15]`

Dependencies

To enable this argument, set the `Sensor` property to an array that supports subarrays and set the `SubarraySteering` property of that array to either `'Phase'` or `'Time'`

Data Types: `double`

Output Arguments

`Y` — Radiated signals

complex-valued M-by-L matrix | complex-valued 1-by-L cell array of structures

Radiated signals, specified as a complex-valued M-by-L matrix or a 1-by-L cell array, where L is the number of radiating angles, `ANG`. M is the length of the input signal, `X`.

- If the Polarization property value is set to `'None'`, the output argument `Y` is an *M*-by-*L* matrix.
- If the Polarization property value is set to `'Combined'` or `'Dual'`, `Y` is a 1-by-*L* cell array of structures. Each cell corresponds to a separate radiating signal. Each `struct` contains three column vectors containing the *X*, *Y*, and *Z* components of the polarized fields defined with respect to the global coordinate system.

Data Types: `double`

Object Functions

To use an object function, specify the System object as the first input argument. For example, to release system resources of a System object named `obj`, use this syntax:

```
release(obj)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
reset	Reset internal states of System object

Examples

Radiate Wideband Energy from Array

Create a 5-by-5 URA and space the elements one-half wavelength apart. The wavelength corresponds to a design frequency of 300 MHz.

Create 5-by-5 URA Array of Cosine Elements

```
c = physconst('LightSpeed');
fc = 100e6;
lam = c/fc;
antenna = phased.CosineAntennaElement('CosinePower',[2,2]);
array = phased.URA('Element',antenna,'Size',[5,5],'ElementSpacing',[0.5,0.5]*lam);
```

Create and Radiate Wideband Signal

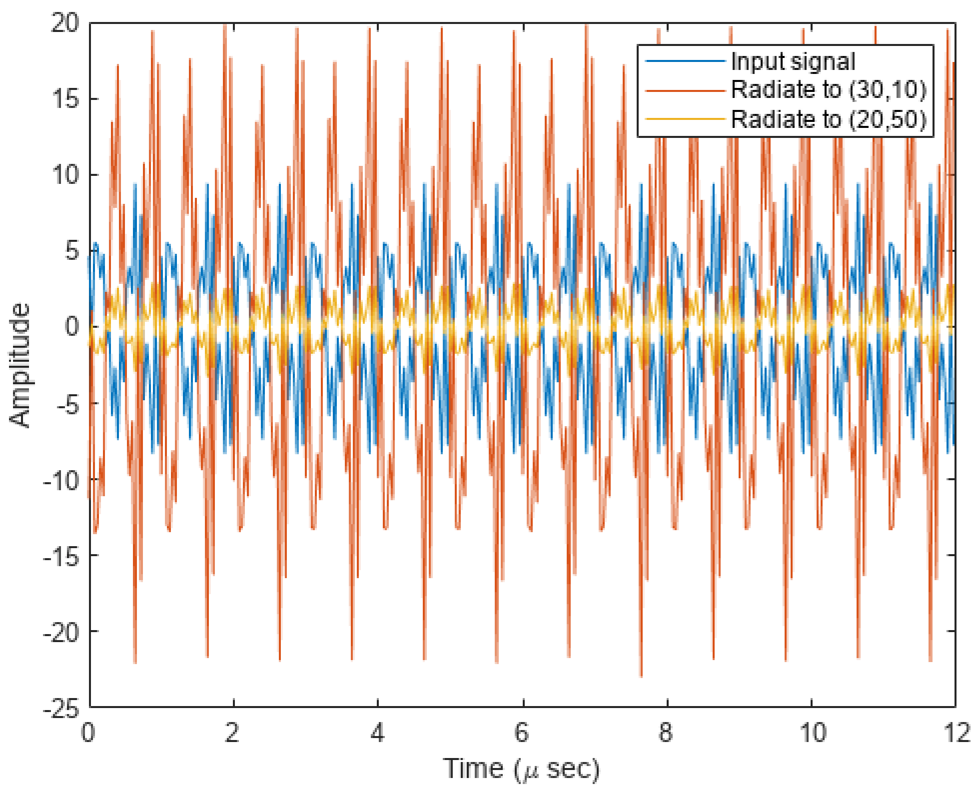
Radiate a wideband signal consisting of three sinusoids at 2, 10 and 11 MHz. Set the sampling rate to 25 MHz. Radiate the fields into two directions: (30,10) degrees azimuth and elevation and (20,50) degrees azimuth and elevation.

```
fs = 25e6;
f1 = 2e6;
f2 = 10e6;
f3 = 11e6;
dt = 1/fs;
Tsig = 100e-6;
t = [0:dt:Tsig];
sig = 5.0*sin(2*pi*f1*t) + 2.0*sin(2*pi*f2*t + pi/10) + 4*sin(2*pi*f3*t + pi/2);
radiatingangles = [30 10; 20 50]';
radiator = phased.WidebandRadiator('Sensor',array,'CarrierFrequency',fc,'SampleRate',fs);
radsig = radiator(sig.',radiatingangles);
```

Plot Radiated Signal

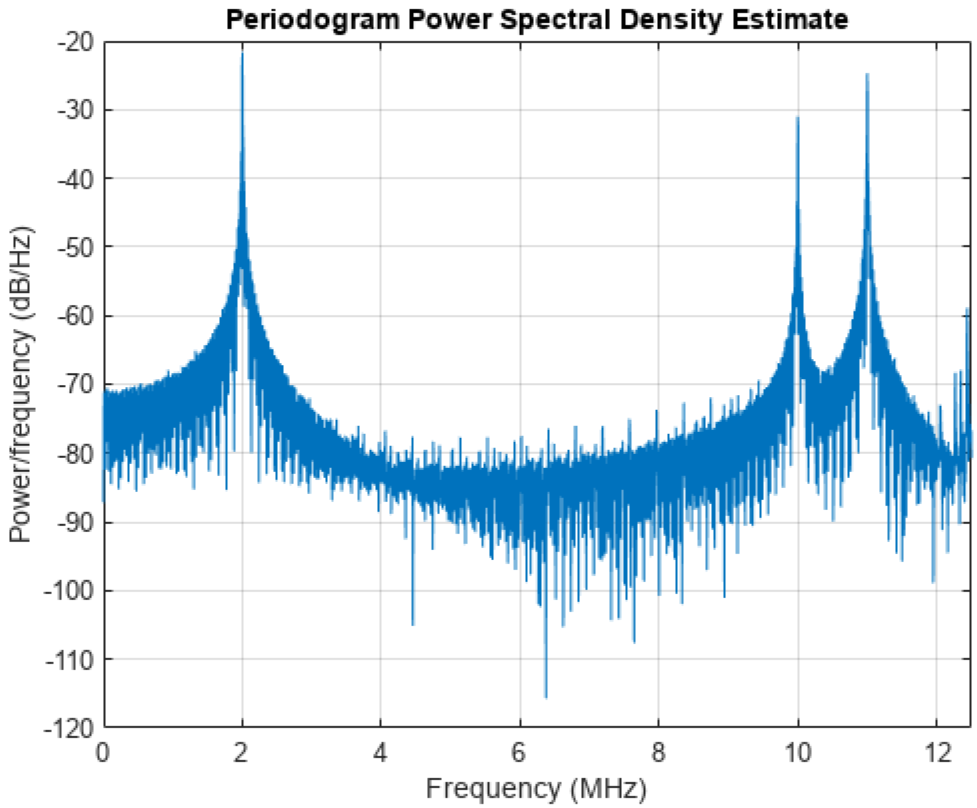
Plot the input signal to the radiator and the radiated signals.

```
plot(t(1:300)*1e6,real(sig(1:300)))
hold on
plot(t(1:300)*1e6,real(radsig(1:300,1)))
plot(t(1:300)*1e6,real(radsig(1:300,2)))
hold off
xlabel('Time (\mu sec)')
ylabel('Amplitude')
legend('Input signal','Radiate to (30,10)','Radiate to (20,50)')
```



Plot the spectra of the signal that is radiated to (30,10) degrees.

```
periodogram(real(radsig(:,1)),rectwin(size(radsig,1)),4096,fs);
```



Radiate Wideband Polarized Fields from Array

Examine the polarized field produced by the wideband radiator from a five-element uniform line array (ULA) composed of short-dipole antenna elements.

Set up the ULA of five short-dipole antennas with polarization enabled. The element spacing is set to 1/2 wavelength of the carrier frequency. Construct the wideband radiator System object™.

```
fc = 100e6;
c = physconst('LightSpeed');
lam = c/fc;
antenna = phased.ShortDipoleAntennaElement;
array = phased.ULA('Element',antenna,'NumElements',5,'ElementSpacing',lam/2);
```

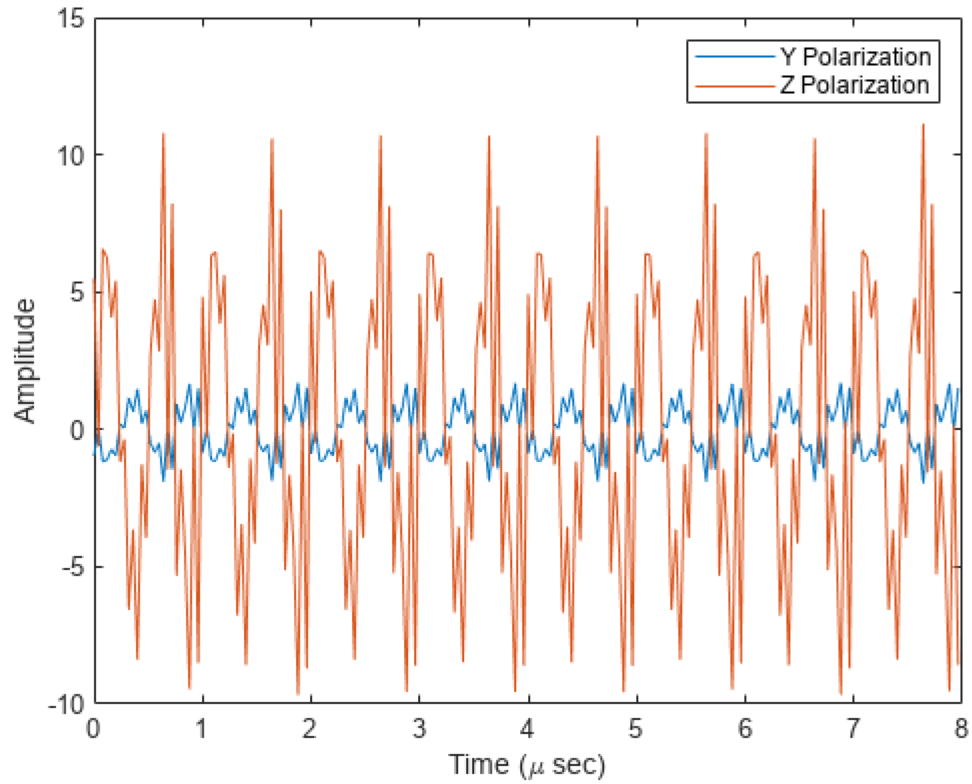
Radiate a signal consisting of the sum of three sine waves. Radiate the signal into two different directions. Radiated angles are azimuth and elevation angles defined with respect to a local coordinate system. The local coordinate system is defined by 10 degree rotation around the x-axis from the global coordinates.

```
fs = 25e6;
f1 = 2e6;
f2 = 10e6;
f3 = 11e6;
dt = 1/fs;
fc = 100e6;
```

```
t = [0:dt:100e-6];
sig = 5.0*sin(2*pi*f1*t) + 2.0*sin(2*pi*f2*t + pi/10) + 4*sin(2*pi*f3*t + pi/2);
radiatingAngle = [30 30; 0 20];
laxes = rotx(10);
radiator = phased.WidebandRadiator('Sensor',array,'SampleRate',fs,...
    'CarrierFrequency',fc,'Polarization','Combined');
y = radiator(sig.',radiatingAngle,laxes);
```

Plot the first 200 samples of the y and z components of the polarized field propagating in the `[30,0]` direction.

```
plot(10^6*t(1:200),real(y(1).Y(1:200)))
hold on
plot(10^6*t(1:200),real(y(1).Z(1:200)))
hold off
xlabel('Time (\mu sec)')
ylabel('Amplitude')
legend('Y Polarization','Z Polarization')
```



More About

Subband Frequency Processing

Subband processing decomposes a wideband signal into multiple subbands and applies narrowband processing to the signal in each subband. The signals for all subbands are summed to form the output signal.

When using wideband frequency System objects or blocks, you specify the number of subbands, N_B, in which to decompose the wideband signal. Subband center frequencies and widths are automatically computed from the total bandwidth and number of subbands. The total frequency band is centered on the carrier or operating frequency, f_c. The overall bandwidth is given by the sample rate, f_s. Frequency subband widths are $\Delta f = f_s/N_B$. The center frequencies of the subbands are

$$f_m = \begin{cases} f_c - \frac{f_s}{2} + (m-1)\Delta f, & N_B \text{ even} \\ f_c - \frac{(N_B-1)f_s}{2N_B} + (m-1)\Delta f, & N_B \text{ odd} \end{cases}, \quad m = 1, \ldots, N_B$$

Some System objects let you obtain the subband center frequencies as output when you run the object. The returned subband frequencies are ordered consistently with the ordering of the discrete Fourier transform. Frequencies above the carrier appear first, followed by frequencies below the carrier.

Version History

Introduced in R2015b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`phased.Radiator` | `phased.Collector` | `phased.WidebandCollector` | `phased.WidebandFreeSpace`

reset

System object: `phased.WidebandRadiator`
Package: `phased`

Reset states of System object

Syntax

`reset(sWBR)`

Description

`reset(sWBR)` resets the internal state of the `phased.WidebandRadiator` object, `sWBR`. If the `SeedSource` property applies, and has the value `'Property'`, then this method resets the state of the random number generator.

Input Arguments

`sWBR` — Wideband radiator
System object

Wideband radiator, specified as a System object.

Example: `phased.WidebandRadiator`

Version History

Introduced in R2015b

step

System object: `phased.WidebandRadiator`
Package: `phased`

Radiate wideband signals

Syntax

```
sigrad = step(sWBR,sig,ang)
sigrad = step(sWBR,sig,ang,laxes)
sigrad = step(sWBR,sig,ang,wts)
sigrad = step(sWBR,sig,ang,steerang)
```

Description

Note Starting in R2016b, instead of using the `step` method to perform the operation defined by the System object, you can call the object with arguments, as if it were a function. For example, `y = step(obj,x)` and `y = obj(x)` perform equivalent operations.

`sigrad = step(sWBR,sig,ang)` radiates the signal `sig` in the directions specified by `ang`. For each direction, the method computes the radiated signal, `sigrad`, by summing the contributions of each element or subarray.

`sigrad = step(sWBR,sig,ang,laxes)` radiates the signal using the specified the local coordinate system of the radiator, `laxes`. This syntax applies when you set the `EnablePolarization` property to `true`.

`sigrad = step(sWBR,sig,ang,wts)` radiates the signal using `wts` as the weight vector when the `WeightsInputPort` property is `true`.

`sigrad = step(sWBR,sig,ang,steerang)` radiates the signal and uses `steerang` as the subarray steering angle. `steerang` must be a length-2 column vector in the form of `[AzimuthAngle; ElevationAngle]`. This syntax applies when you use a subarray as the `Sensor` property and set the `SubarraySteering` property of the sensor to `'Phase'` or `'Time'`.

You can combine optional input arguments when you set their enabling properties in the System object during construction. Optional inputs must be listed in the same order as their enabling properties. For example, `sigrad = step(sWBR,sig,laxes,wts,steerang)` is valid when you set both `EnablePolarization` and `WeightsInputPort` to `true` and set the `SubarraySteering` property of the sensor.

Note The object performs an initialization the first time the object is executed. This initialization locks nontunable properties and input specifications, such as dimensions, complexity, and data type of the input data. If you change a nontunable property or an input specification, the System object issues an error. To change nontunable properties or inputs, you must first call the `release` method to unlock the object.

Input Arguments

sWBR — Wideband radiator
System object

Wideband radiator, specified as a `phased.WidebandRadiator` System object.

Example: `phased.WidebandRadiator`

sig — Input signals
M-by-1 complex-valued column vector | *M*-by-*N* complex-valued matrix

Input signals, specified as an *M*-by-1 complex-valued column vector or *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values (snapshots) of the signal. If `sig` is a column vector, the same signal is radiated through all elements. If `sig` is a matrix, *N* is the number of sensor elements in the array. For subarrays, *N* is the number of subarrays. Each column of `sig` represents the field radiated by the corresponding element or subarray.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[[0;1;2;3;4;3;2;1;0],[1;2;3;4;3;2;1;0;0]]`

Data Types: `double`
Complex Number Support: Yes

ang — Radiating directions
2-by-*L* real-valued matrix | 1-by-*L* real-valued row vector

Radiating directions of the signal, specified as 2-by-*L* real-valued matrix or 1-by-*L* real-valued row vector. The quantity *L* is the number of directions to radiate. If `ang` is a matrix, each column has the form `[azimuth;elevation]`. If `ang` is a row vector, each entry represents the azimuthal direction. The elevation direction is zero degrees. Angle units are in degrees. Angles are defined with respect to the local coordinate system of the array.

When the sensory array is a uniform linear array, `ang` represents the broadside angle.

Data Types: `double`

laxes — Local coordinate system axes
`eye(3,3)` (default) | 3-by-3 real-valued orthonormal matrix

Local coordinate system axes, specified as a 3-by-3 real-valued matrix orthonormal matrix. The matrix columns specify the *x*, *y*, and *z* axes of the local coordinate system. Each column takes the form `[x;y;z]` with respect to the global coordinate system. This argument only applies when the `EnablePolarization` property is set to `true`.

Data Types: `double`

wts — Weight vector
`ones(N,1)` (default) | *N*-by-1 complex-valued column vector

Weight vector, specified as an *N*-by-1 complex-valued column vector. Each weight vector element multiplies the signal at the corresponding element or subarray. *N* is the number of radiating elements or subarrays. This argument only applies when the `WeightsInputPort` property is `true`.

Data Types: `double`
Complex Number Support: Yes

`steerang` — Subarray steering angle
2-by-1 real-valued column vector

Subarray steering angle, specified as a 2-by-1 real-valued column vector in the form of `[AzimuthAngle; ElevationAngle]`. This argument applies only when the `Sensor` property refers to a subarray and the `SubarraySteering` property of the sensor is set to `'Phased'` or `'Time'`. Angles are defined with respect to the local coordinate system axes. Angle units are in degrees.

Data Types: `double`

Output Arguments

`sigrad` — Radiated signal
M-by-*L* complex-valued matrix | 1-by-*L* array of `struct` type

Radiated signal, returned as an *M*-by-*L* complex-valued matrix or 1-by-*L* array of `struct` type depending on whether polarization is enabled. The radiated field is the combined far-field output from all elements or subarrays. The quantity *M* is the number of sample values (snapshots) of the signal. The quantity *L* is the number of entries in `ang`.

- If you set `EnablePolarization` to `false`, `sigrad` is an *M*-by-*L* complex-valued matrix.
- If you set `EnablePolarization` is `true`, `sigrad` is a 1-by-*L* array of `struct` type. Each `struct` in the array has three data fields: `sigrad.X`, `sigrad.Y`, `sigrad.Z` which correspond to the *x*, *y*, and *z* components of the electromagnetic field. Electromagnetic field components are defined with respect to the global coordinate system. Each data field is an *M*-by-1 column vector.

Examples

Radiate Wideband Energy from Array

Create a 5-by-5 URA and space the elements one-half wavelength apart. The wavelength corresponds to a design frequency of 300 MHz.

Create 5-by-5 URA Array of Cosine Elements

```
c = physconst('LightSpeed');
fc = 100e6;
lam = c/fc;
antenna = phased.CosineAntennaElement('CosinePower',[2,2]);
array = phased.URA('Element',antenna,'Size',[5,5],'ElementSpacing',[0.5,0.5]*lam);
```

Create and Radiate Wideband Signal

Radiate a wideband signal consisting of three sinusoids at 2, 10 and 11 MHz. Set the sampling rate to 25 MHz. Radiate the fields into two directions: (30,10) degrees azimuth and elevation and (20,50) degrees azimuth and elevation.

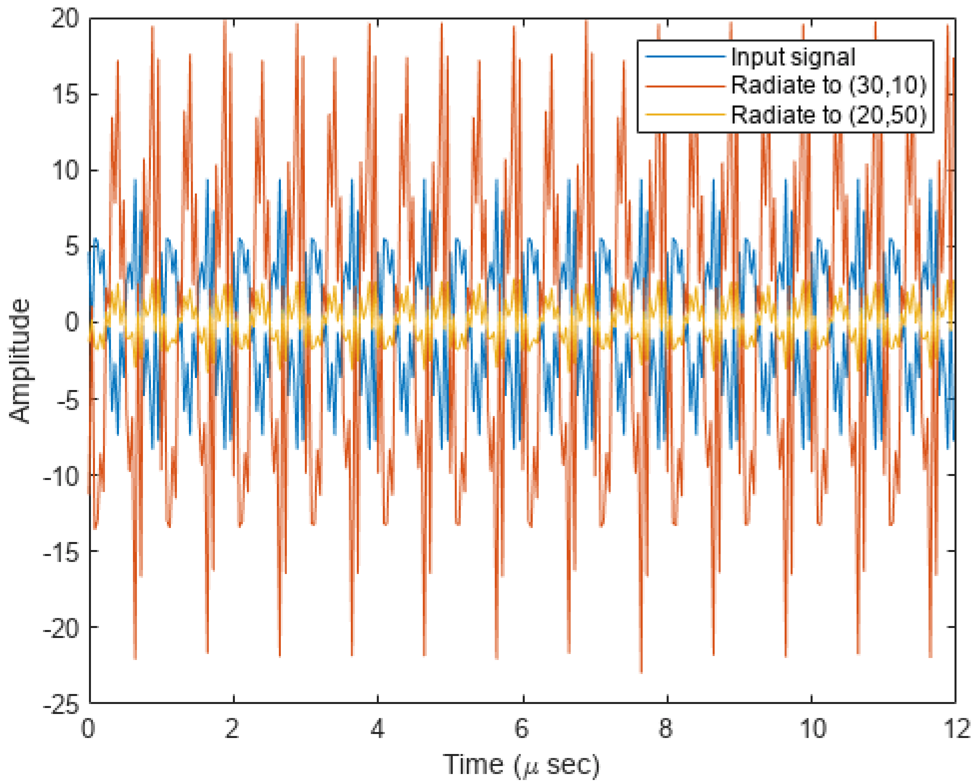
```
fs = 25e6;
f1 = 2e6;
f2 = 10e6;
```

```
f3 = 11e6;
dt = 1/fs;
Tsig = 100e-6;
t = [0:dt:Tsig];
sig = 5.0*sin(2*pi*f1*t) + 2.0*sin(2*pi*f2*t + pi/10) + 4*sin(2*pi*f3*t + pi/2);
radiatingangles = [30 10; 20 50]';
radiator = phased.WidebandRadiator('Sensor',array,'CarrierFrequency',fc,'SampleRate',fs);
radsig = radiator(sig.',radiatingangles);
```

Plot Radiated Signal

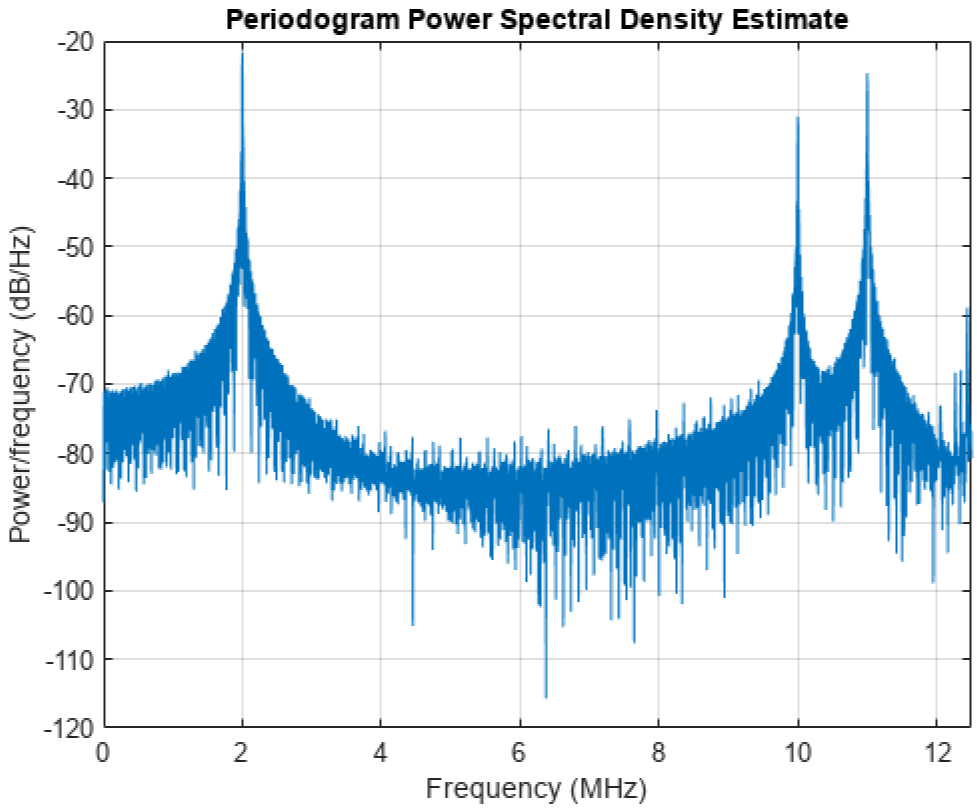
Plot the input signal to the radiator and the radiated signals.

```
plot(t(1:300)*1e6,real(sig(1:300)))
hold on
plot(t(1:300)*1e6,real(radsig(1:300,1)))
plot(t(1:300)*1e6,real(radsig(1:300,2)))
hold off
xlabel('Time (\mu sec)')
ylabel('Amplitude')
legend('Input signal','Radiate to (30,10)','Radiate to (20,50)')
```



Plot the spectra of the signal that is radiated to (30,10) degrees.

```
periodogram(real(radsig(:,1)),rectwin(size(radsig,1)),4096,fs);
```



Radiate Wideband Polarized Fields from Array

Examine the polarized field produced by the wideband radiator from a five-element uniform line array (ULA) composed of short-dipole antenna elements.

Set up the ULA of five short-dipole antennas with polarization enabled. The element spacing is set to 1/2 wavelength of the carrier frequency. Construct the wideband radiator System object™.

```
fc = 100e6;
c = physconst('LightSpeed');
lam = c/fc;
antenna = phased.ShortDipoleAntennaElement;
array = phased.ULA('Element',antenna,'NumElements',5,'ElementSpacing',lam/2);
```

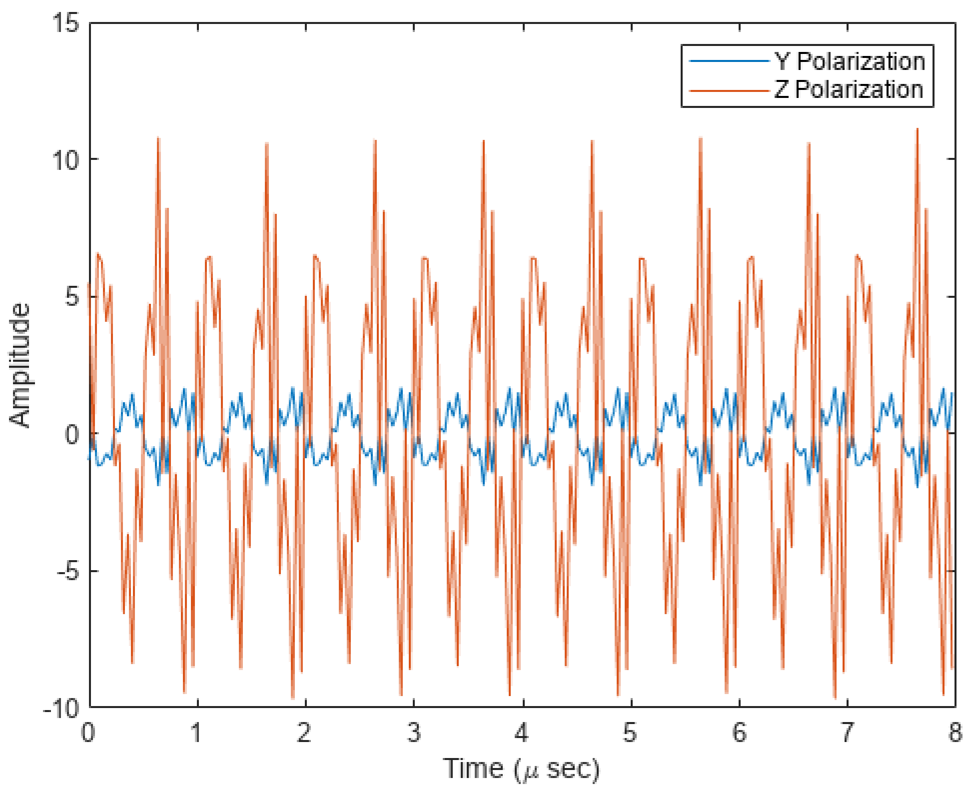
Radiate a signal consisting of the sum of three sine waves. Radiate the signal into two different directions. Radiated angles are azimuth and elevation angles defined with respect to a local coordinate system. The local coordinate system is defined by 10 degree rotation around the x-axis from the global coordinates.

```
fs = 25e6;
f1 = 2e6;
f2 = 10e6;
f3 = 11e6;
dt = 1/fs;
fc = 100e6;
```

```
t = [0:dt:100e-6];
sig = 5.0*sin(2*pi*f1*t) + 2.0*sin(2*pi*f2*t + pi/10) + 4*sin(2*pi*f3*t + pi/2);
radiatingAngle = [30 30; 0 20];
laxes = rotx(10);
radiator = phased.WidebandRadiator('Sensor',array,'SampleRate',fs,...
    'CarrierFrequency',fc,'Polarization','Combined');
y = radiator(sig.',radiatingAngle,laxes);
```

Plot the first 200 samples of the *y* and *z* components of the polarized field propagating in the `[30,0]` direction.

```
plot(10^6*t(1:200),real(y(1).Y(1:200)))
hold on
plot(10^6*t(1:200),real(y(1).Z(1:200)))
hold off
xlabel('Time (\mu sec)')
ylabel('Amplitude')
legend('Y Polarization','Z Polarization')
```



Version History

Introduced in R2015b

See Also

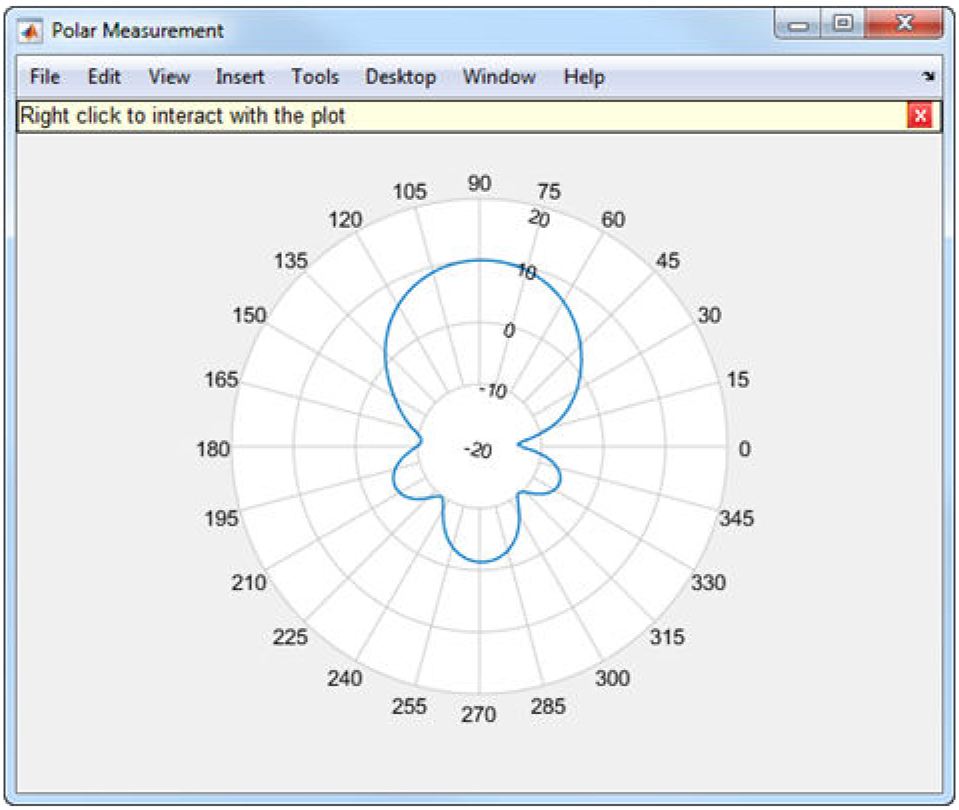
phased.BeamscanEstimator | phased.Collector | phased.WidebandCollector | phased.Radiator | phased.RootMUSICEstimator

polarpattern

Interactive plot of radiation patterns in polar format

Description

The `polarpattern` object creates an interactive plot of antenna or array radiation patterns in polar format with uniformly spaced angles. You can also plot other types of polar data. Use this plot for interactive data visualization or measurement. To change the properties, zoom in, or add more data to the plot, right-click or scroll or drag the **Polar Measurement** window.



Creation

Syntax

```
polarpattern
polarpattern(data)
polarpattern(angle,magnitude)
polarpattern( ___ ,Name,Value)
polarpattern(ax, ___ )
p = polarpattern( ___ )
```

```
p = polarpattern('gco')
```

Description

`polarpattern` creates an empty polar plot. You can add plots of antenna or array radiation patterns and other types of data to the plot by importing saved polari objects from MAT-files.

`polarpattern(data)` creates a polar plot with real magnitude values in the vector `data` with angles uniformly spaced on the unit circle starting at `0` degrees. Magnitudes may be negative when dB data units are used. For a matrix `data`, columns of `data` are independent datasets. For *N*-`data` arrays, dimensions 2 and greater are independent datasets. For complex values, magnitude and angle are derived from `data`.

`polarpattern(angle,magnitude)` creates a polar plot for a set of angles and corresponding magnitudes. You can also create polar plots from multiple sets of angle vectors in degrees and corresponding sets of magnitude using the syntax: `polarpattern(angle1, magnitude1,..., angleN, magnitudeN)`.

`polarpattern( ___ ,Name,Value)` creates a polar plot, with additional properties specified by one or more name-value pair arguments. `Name` is the property name and `Value` is the corresponding property value. You can specify several name-value pair arguments in any order as `Name1`, `Value1`, `...`, `NameN`, `ValueN`. Unspecified properties retain their default values. To list all the property names and values, use `details(p)`. You can use the properties to extract data about the radiation pattern from the polar plot. For example, `p = polarpattern(data,'Peaks',3)` identifies and displays the three highest peaks in the pattern data.

For a list of properties, see PolarPattern.

`polarpattern(ax, ___ )` creates a polar plot using axes object, `ax` instead of the current axes object.

`p = polarpattern( ___ )` creates a polari object using any combination of input arguments from the previous syntaxes. Use this syntax to customize the plot or add measurements.

`p = polarpattern('gco')` creates a polar plot object from the polar pattern in the current figure.

Input Arguments

`data` — Antenna or array data

real length-*M* vector | real *M*-by-*N* matrix | real multidimensional array | complex vector or matrix

Antenna or array data, specified as one of these options

- A real length-*M* vector, containing *M* magnitude values with angles their defined as $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees.
- A real *M*-by-*N* matrix, containing *M* magnitude values in a dataset and *N* such independent data sets. Each column of the matrix has angles in degrees from the vector $\frac{(0:M-1)}{M} \times 360^{\circ}$.
- A real multidimensional array. Arrays with `2` or more dimensions contain independent data sets.
- A complex vector or matrix, that contains Cartesian coordinates (x, y) of each point. x contains the real part of the `data` and y contains the imaginary part of the `data`.

When the data is in a logarithmic form, such as dB, magnitude values can be negative. In this case, `polarpattern` plots the smallest magnitude values at the origin of the polar plot and largest magnitude values at the maximum radius.

Data Types: `double`

`angle` — Set of angles
vector

Set of angles in degrees, specified as a vector.

Data Types: `double`

`magnitude` — Set of magnitude values
vector | matrix

Set of magnitude values, specified as a vector or a matrix. If you specify this input as a matrix, each column is an independent set of magnitude values and corresponds to the same set of angles in the same column of the angle input .

Data Types: `double`

`ax` — Axes of polar plot
`axes` object

Axes of the polar plot, specified as an axes object.

Output Arguments

`p` — polari object
`polari` object

Stores a polari object with a set of properties. Use p to modify properties of the plot after creation. For a list of all the properties, see PolarPattern Properties (Antenna Toolbox).

Example: `P = polarplot(V)`

Object Functions

Examples

Plot Cosine Pattern in Polar Coordinates

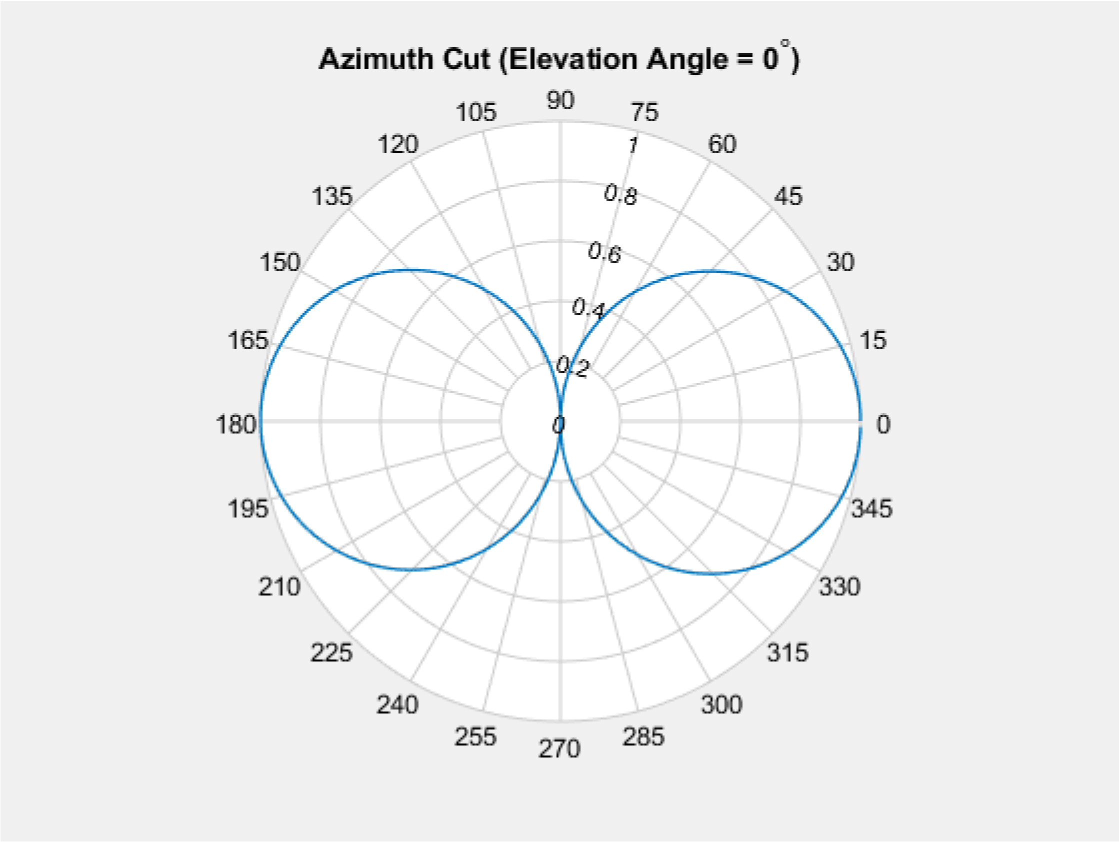
Specify a cosine antenna pattern from 0° to 360° in azimuth at 0° elevation. Then, plot the antenna pattern using `polarpattern`.

Create the pattern.

```
az = [0:360];
p = abs(cosd(az));
```

Plot the polar pattern of the antenna for an azimuth cut at 0° elevation.

```
polarpattern(p,'TitleTopTextInterpreter','tex','TitleTop','Azimuth Cut (Elevation Angle = 0^{\ci
```



Azimuth Pattern of a 3-by-2 URA

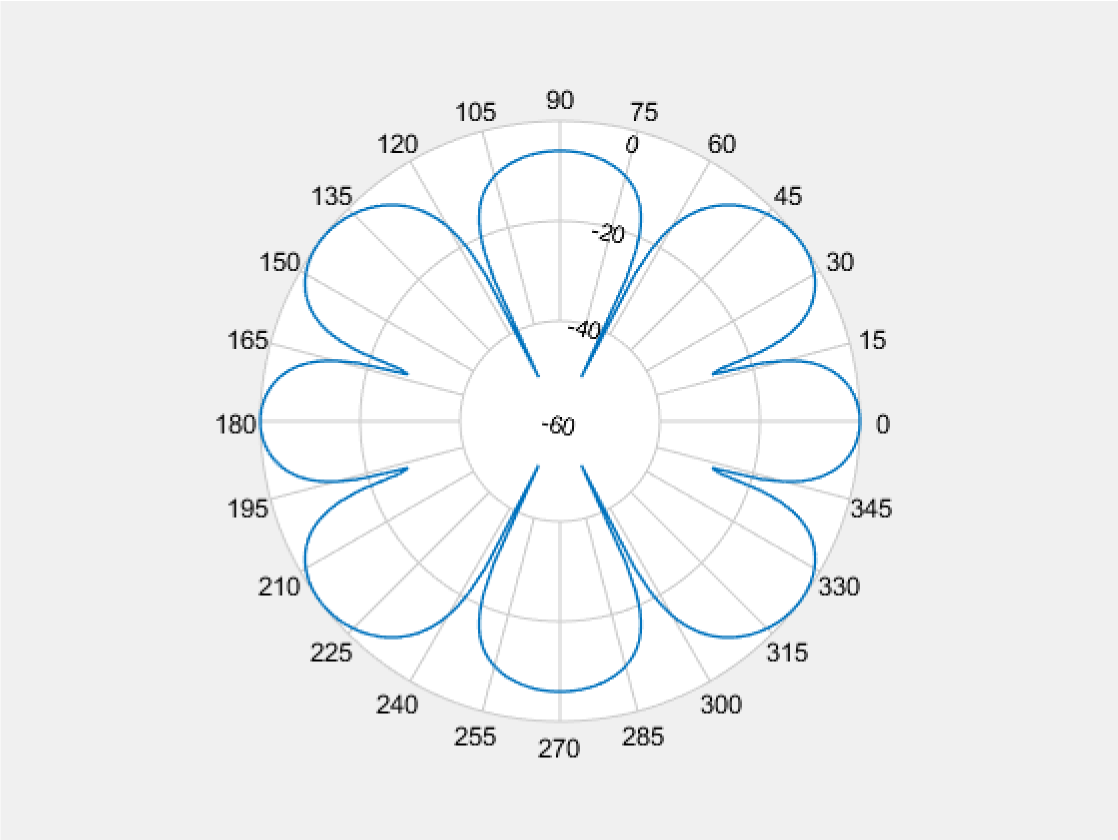
Construct a 3-by-2 rectangular lattice URA. By default, the array consists of isotropic antenna elements. Assume the operating frequency is 1 GHz. Then, plot the antenna pattern using `polarpattern`.

Create the array.

```
array = phased.URA('Size',[3 2]);
fc = 1.0e9;
```

Plot the polar pattern of the array for an elevation cut at 0° azimuth.

```
c = physconst('LightSpeed');
p = pattern(array,fc,[-180:180],0,'PropagationSpeed',c,'CoordinateSystem',...
    'polar','Type','powerdb','Normalize',true);
polarpattern([-180:180],p);
```



Polar Pattern Display with Title for Short-Dipole Antenna

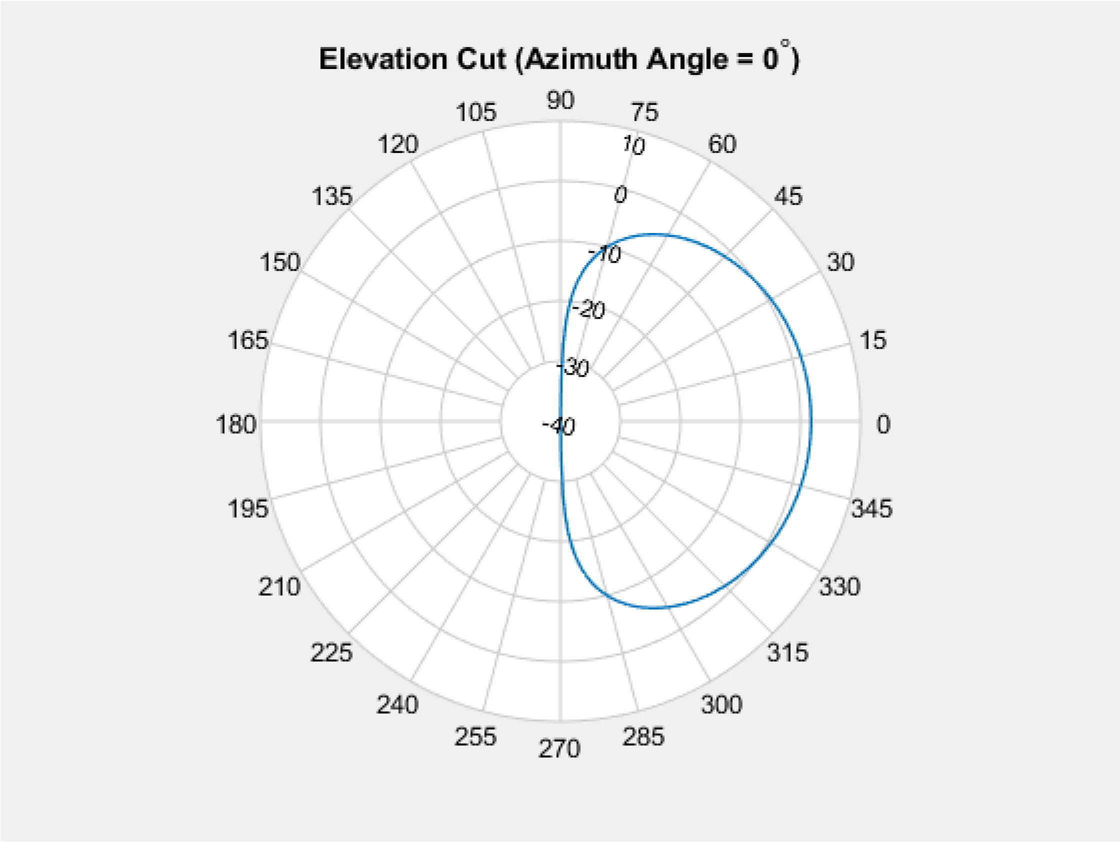
Specify a short-dipole antenna with the dipole oriented along the *z*-axis and operating at 250 MHz. Then, plot the antenna pattern using `polarpattern` and specifying a title.

Create the short-dipole antenna element System object™.

```
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6,600e6],...
    'AxisDirection','Z');
fc = 250.0e6;
```

Plot the polar pattern of the antenna for an elevation cut at 0° azimuth.

```
v = pattern(antenna,fc,0,-90:90);
polarpattern([-90:90],v,'TitleTopTextInterpreter','tex',...
    'TitleTop','Elevation Cut (Azimuth Angle = 0^{\circ})');
```



Polar Pattern Properties

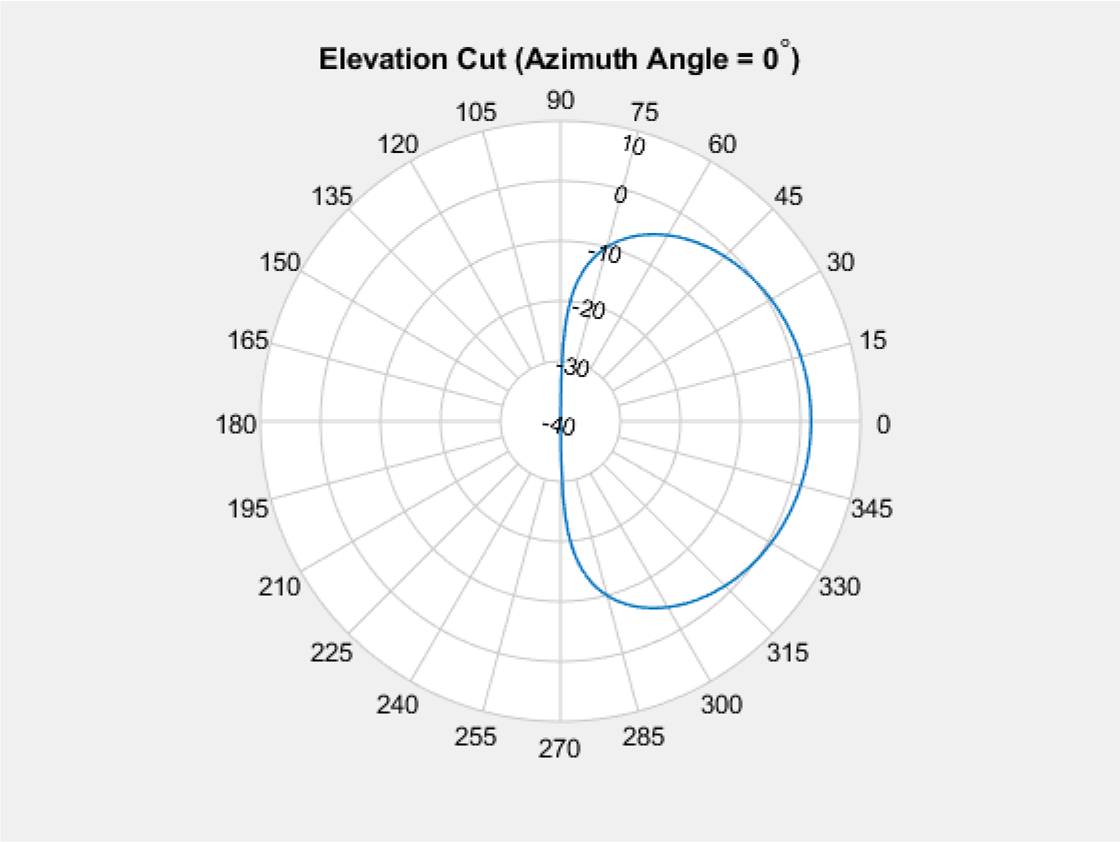
Specify a short-dipole antenna with the dipole oriented along the *z*-axis and operating at 250 MHz. Then, plot the antenna pattern using `polarpattern` and specifying a title.

Create the short-dipole antenna element System object™.

```
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[100e6,600e6],...
    'AxisDirection','Z');
fc = 250.0e6;
```

Create the polar pattern of the antenna for an elevation cut at 0° azimuth.

```
p = pattern(antenna,fc,0,-90:90);
P = polarpattern([-90:90],p,'TitleTopTextInterpreter','tex',...
    'TitleTop','Elevation Cut (Azimuth Angle = 0^{\circ})');
```



Display the properties of the plot.

```
details(P)
```

```
  internal.polari handle with properties:

                 Interactive: 1
                LegendLabels: ''
              AntennaMetrics: 0
                   CleanData: 0
                   AngleData: [181x1 double]
               MagnitudeData: [181x1 double]
               IntensityData: []
                AngleMarkers: [0x1 struct]
               CursorMarkers: [0x1 struct]
                 PeakMarkers: [0x1 struct]
               ActiveDataset: 1
             AngleLimVisible: 0
               LegendVisible: 0
                        Span: 0
                    TitleTop: 'Elevation Cut (Azimuth Angle = 0^{\circ})'
                 TitleBottom: ''
                       Peaks: []
                    FontSize: 10
                MagnitudeLim: [-40 10]
          MagnitudeAxisAngle: 75
               MagnitudeTick: [-40 -30 -20 -10 0 10]
     MagnitudeTickLabelColor: 'k'
```

```
                    AngleLim: [0 360]
              AngleTickLabel: {'0'  '15'  '30'  '45'  '60'  '75'  '90'  '105'  '120'  '135'
         AngleTickLabelColor: 'k'
  TitleTopFontSizeMultiplier: 1.1000
TitleBottomFontSizeMultiplier: 0.9000
          TitleTopFontWeight: 'bold'
       TitleBottomFontWeight: 'normal'
     TitleTopTextInterpreter: 'tex'
  TitleBottomTextInterpreter: 'none'
              TitleTopOffset: 0.1500
           TitleBottomOffset: 0.1500
                    ToolTips: 1
          MagnitudeLimBounds: [-Inf Inf]
 MagnitudeFontSizeMultiplier: 0.9000
     AngleFontSizeMultiplier: 1
                  AngleAtTop: 90
              AngleDirection: 'ccw'
             AngleResolution: 15
      AngleTickLabelRotation: 0
        AngleTickLabelFormat: '360'
     AngleTickLabelColorMode: 'contrast'
                PeaksOptions: {}
       AngleTickLabelVisible: 1
                       Style: 'line'
                   DataUnits: 'dB'
                DisplayUnits: 'dB'
               NormalizeData: 0
            ConnectEndpoints: 0
         DisconnectAngleGaps: 0
                   EdgeColor: 'k'
                   LineStyle: '-'
                   LineWidth: 1
                    FontName: 'Helvetica'
                FontSizeMode: 'auto'
         GridForegroundColor: [0.8000 0.8000 0.8000]
         GridBackgroundColor: 'w'
            DrawGridToOrigin: 0
                GridOverData: 0
          GridAutoRefinement: 0
                   GridWidth: 0.5000
                 GridVisible: 1
                    ClipData: 1
             TemporaryCursor: 1
            MagnitudeLimMode: 'auto'
      MagnitudeAxisAngleMode: 'auto'
           MagnitudeTickMode: 'auto'
 MagnitudeTickLabelColorMode: 'contrast'
   MagnitudeTickLabelVisible: 1
              MagnitudeUnits: ''
              IntensityUnits: ''
                      Marker: 'none'
                  MarkerSize: 6
                      Parent: [1x1 Figure]
                    NextPlot: 'replace'
                  ColorOrder: [7x3 double]
             ColorOrderIndex: 1
                SectorsColor: [16x3 double]
                SectorsAlpha: 0.5000
```

```
         View: 'full'
ZeroAngleLine: 0
```

Version History

Introduced in R2016a

See Also

PolarPattern Properties

add

Add data to polar plot

Syntax

```
add(p,d)
add(p,angle,magnitude)
```

Description

`add(p,d)` adds new antenna data to the polar plot, `p` based on the real amplitude values, `data`.

`add(p,angle,magnitude)` adds data sets of `angle` vectors and corresponding `magnitude` matrices to polar plot `p`.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`data` — Antenna or array data
real length-*M* vector | real *M*-by-*N* matrix | real *N-D* array | complex vector or matrix

Antenna or array data, specified as one of the following:

- A real length-*M* vector, where *M* contains the magnitude values with angles assumed to be $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees.
- A real *M*-by-*N* matrix, where *M* contains the magnitude values and *N* contains the independent data sets. Each column in the matrix has angles taken from the vector $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees. The set of each angle can vary for each column.
- A real *N-D* array, where *N* is the number of dimensions. Arrays with dimensions `2` and greater are independent data sets.
- A complex vector or matrix, where `data` contains Cartesian coordinates (*(x,y)* of each point. *x* contains the real part of `data` and *y* contains the imaginary part of `data`.

When data is in a logarithmic form such as dB, magnitude values can be negative. In this case,`polarpattern` plots the lowest magnitude values at the origin of the polar plot and highest magnitude values at the maximum radius.

`angle` — Set of angles
vector in degrees

Set of angles, specified as a vector in degrees.

magnitude — Set of magnitude values
vector | matrix

Set of magnitude values, specified as a vector or a matrix. For a matrix of magnitude values, each column is an independent set of magnitude values and corresponds to the same set of angles.

Examples

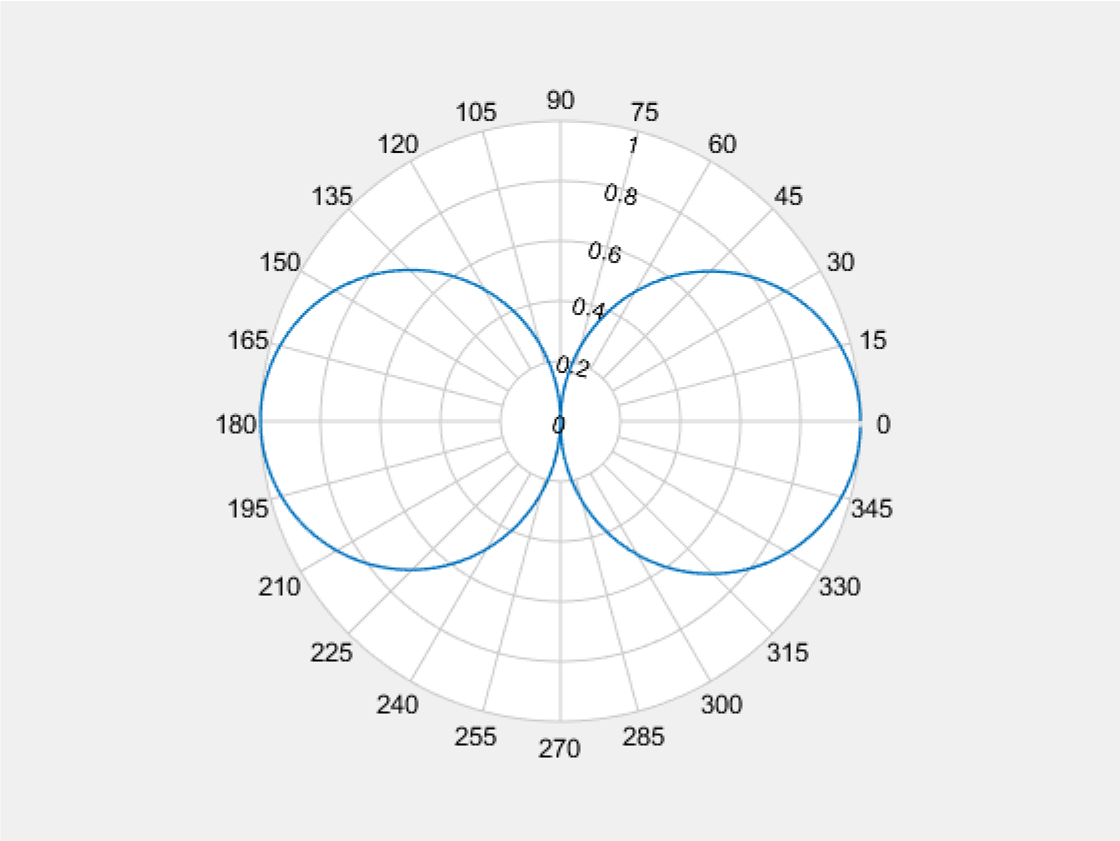
Add Data to Existing Polar Plot

Create a cosine-pattern antenna and plot the pattern from 0° to 36°.

```
az = [0:360];
p1 = abs(cosd(az));
```

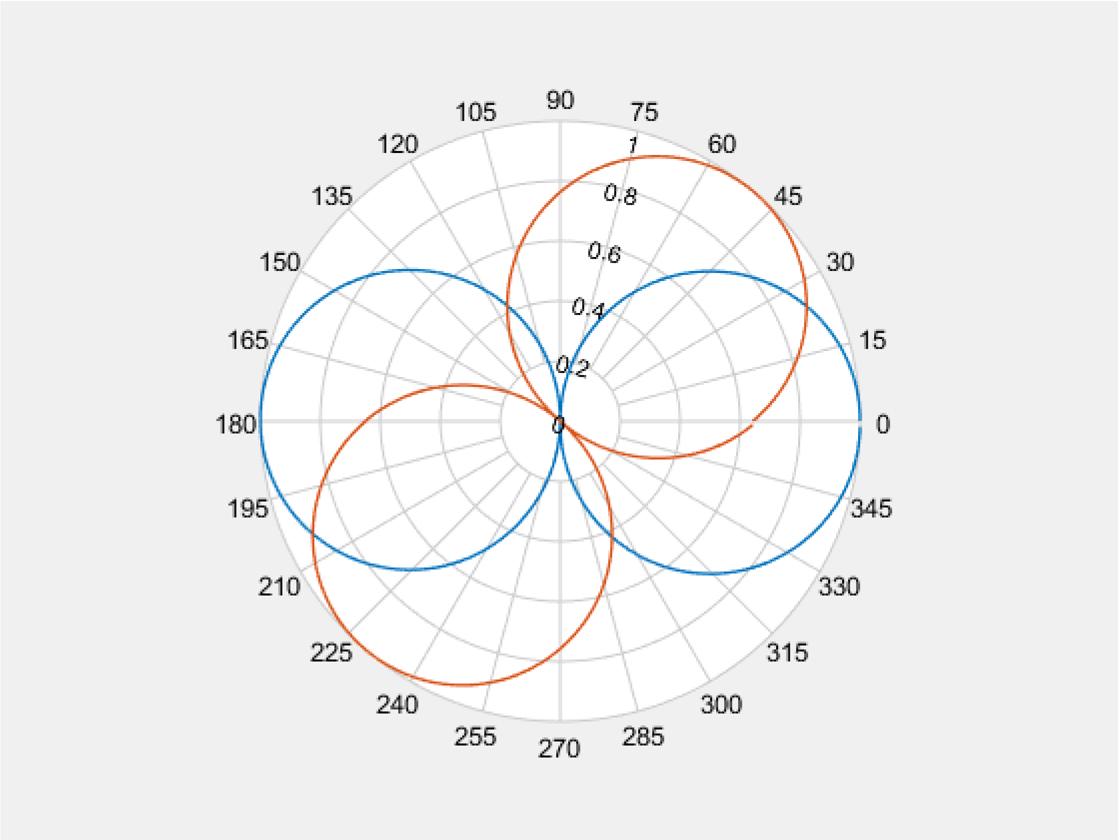
Plot the polar pattern.

```
P = polarpattern(p1);
```



Create a second cosine-pattern antenna rotated by 60°. Add this pattern to the existing pattern.

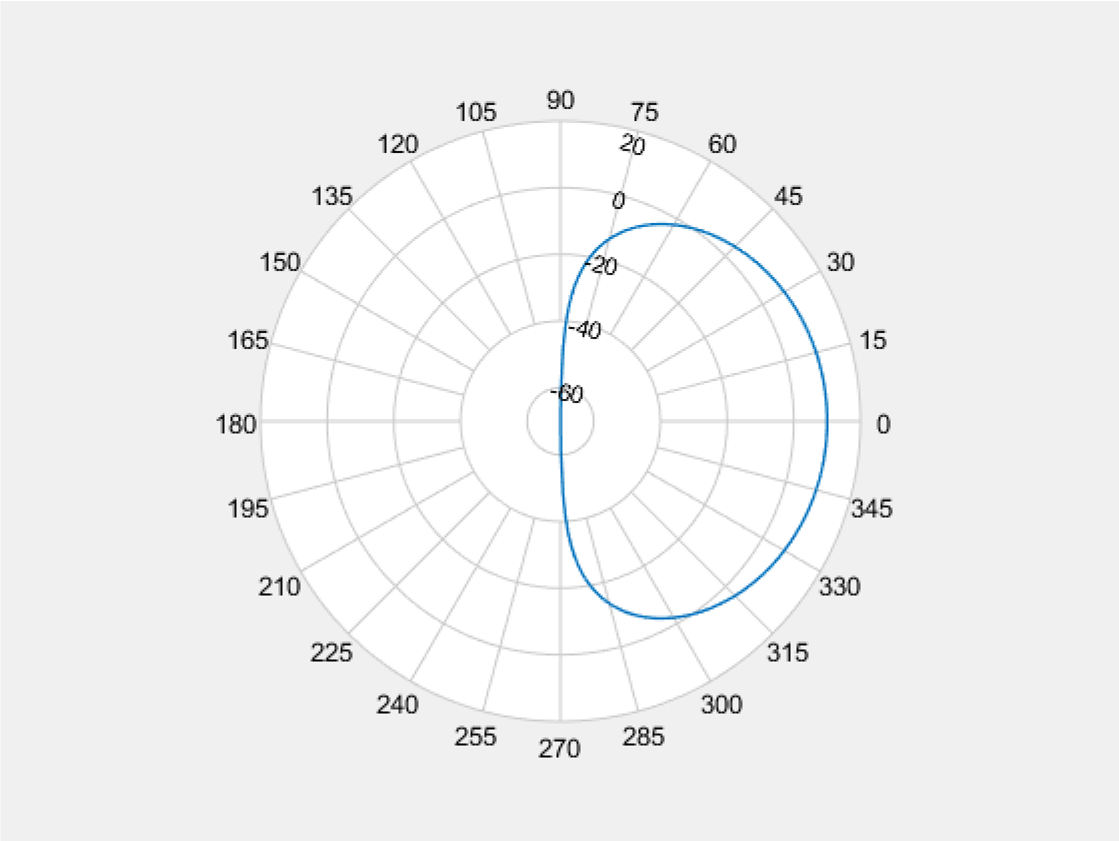
```
p2 = abs(cosd(az - 50));
add(P,p2);
```



Add Second Plot to Polar Pattern

Create a cosine antenna and plot the polar pattern of its directivity at 75 MHz.

```
cosineantenna = phased.CosineAntennaElement('FrequencyRange',[1.0e0 100.0e9],...
    'CosinePower',[2,2]);
p1 = pattern(cosineantenna,75.0e6,[-90:90],0,'Type','Directivity');
P = polarpattern([-90:90],p1);
```



Create an isotropic antenna. Calculate the directivity of this antenna at 75 MHz.

```
isoantenna = phased.IsotropicAntennaElement('FrequencyRange',...
    [1.0e0 100.0e9]);
p2 = pattern(isoantenna,75.0e6,[-180:180],0,'Type','Directivity');
```

Add the directivity plot of the isotropic antenna to the directivity plot of the cosine antenna.

```
add(P,[-180:180],p2);
```

Version History

Introduced in R2016a

See Also

`addCursor` | `animate` | `createLabels` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

addCursor

Add cursor to polar plot angle

Syntax

```
addCursor(p,angle)
addCursor(p,angle,index)
id = addCursor( ___ )
```

Description

`addCursor(p,angle)` adds a cursor to the active polar plot, `p`, at the data point closest to the specified `angle`. Angle units are in degrees.

The first cursor added is called `'C1'`, the second `'C2'`, and so on.

`addCursor(p,angle,index)` adds a cursor at a specified data set `index`. `index` can be a vector of indices.

`id = addCursor( ___ )` returns a cell array with one ID for each cursor created. You can specify any of the arguments from the previous syntaxes.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

angle — Angle values
scalar in degrees | vector in degrees

Angle values at which the cursor is added, specified as a scalar or a vector in degrees.

index — Data set index
scalar | vector

Data set index, specified as a scalar or a vector.

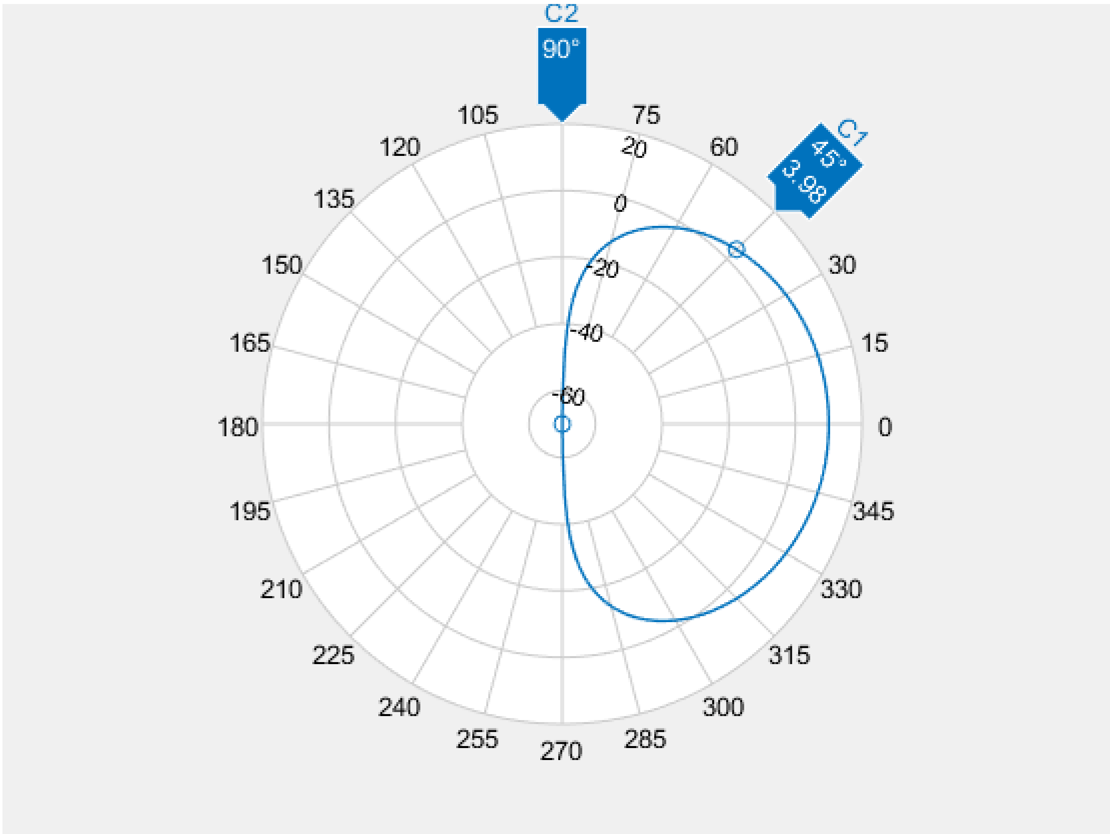
Examples

Add Cursors to Single Polar Pattern Plot

Create a cosine antenna and plot the polar pattern of its directivity at 75 MHz. Then add cursors at two 150° and 270°.

```
cosineantenna = phased.CosineAntennaElement('FrequencyRange',[1.0e0 100.0e9],...
    'CosinePower',[2,2]);
p = pattern(cosineantenna,75.0e6,[-90:90],0,'Type','Directivity');
```

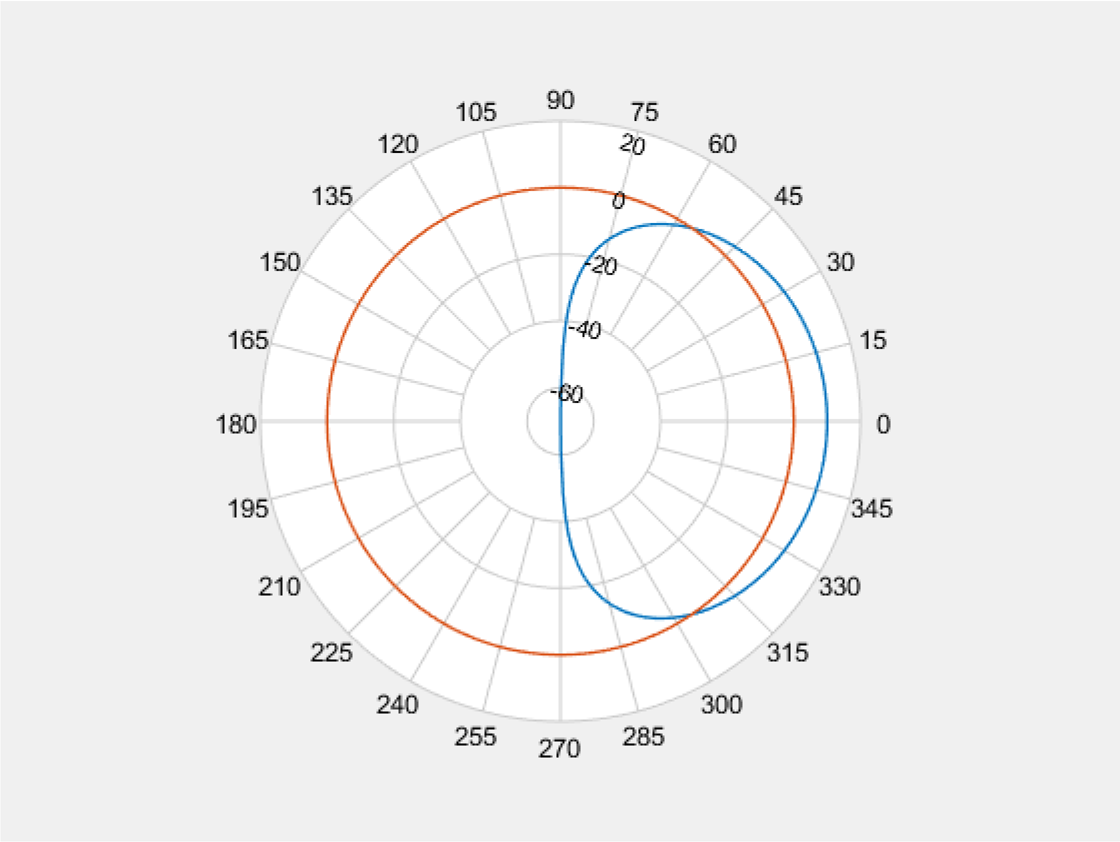
```
P = polarpattern([-90:90],p);
addCursor(P,[45 135]);
```



Add Cursors to Multiple Polar Pattern Plots

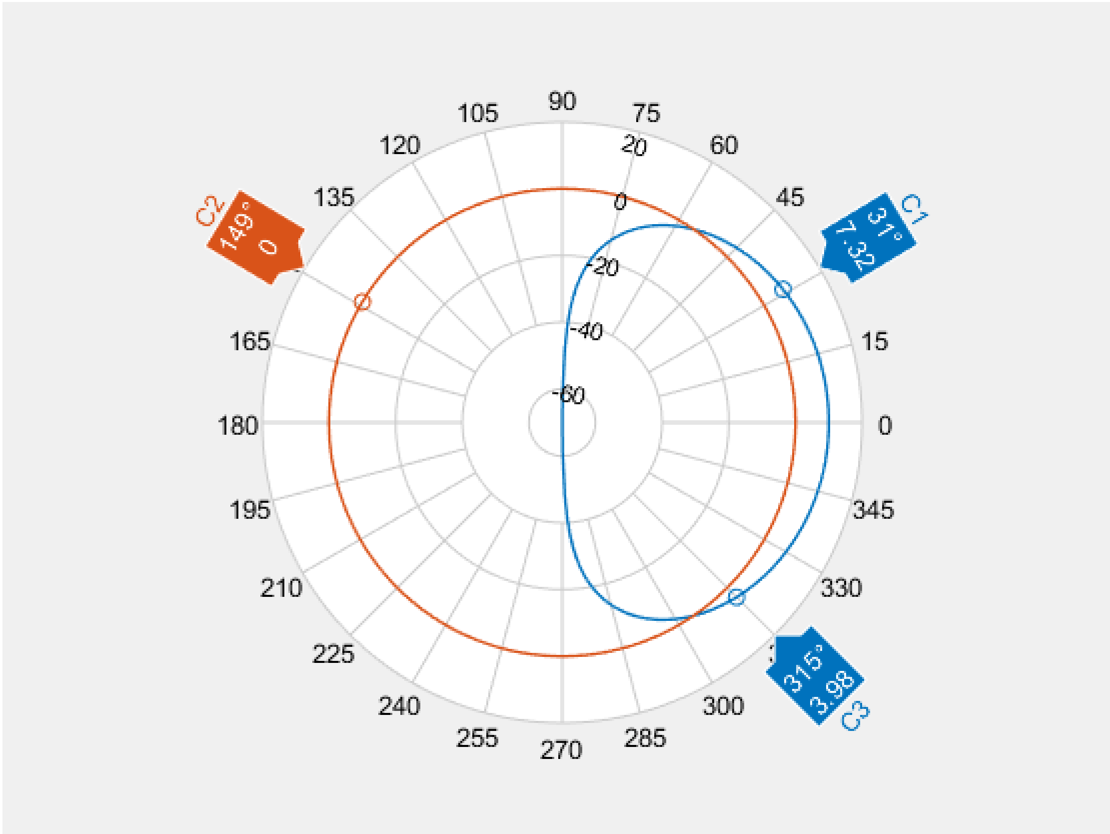
Create a cosine antenna and plot the polar pattern of its directivity at 75 MHz. Then create an isotropic antenna. Also calculate the directivity of this antenna at 75 MHz. Add the directivity plot of the isotropic antenna to the directivity plot of the cosine antenna. Then add cursors at several points.

```
cosineantenna = phased.CosineAntennaElement('FrequencyRange',[1.0e0 100.0e9],...
    'CosinePower',[2,2]);
p1 = pattern(cosineantenna,75.0e6,[-90:90],0,'Type','Directivity');
P = polarpattern([-90:90],p1);
isoantenna = phased.IsotropicAntennaElement('FrequencyRange',...
    [1.0e0 100.0e9]);
p2 = pattern(isoantenna,75.0e6,[-180:180],0,'Type','Directivity');
add(P,[-180:180],p2);
```



Add a cursor at approximately 30° to the cosine antenna pattern (designated by index 1) and at 150° and 270° to the isotropic polar pattern (designated by index 2).

```
addCursor(P,[30.5 149.0 314.7],[1 2 1]);
```



Version History

Introduced in R2016a

See Also

`add` | `animate` | `createLabels` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

animate

Replace existing data with new data for animation

Syntax

```
animate(p,data)
animate(p,angle,magnitude)
```

Description

`animate(p,data)` removes all the current data from polar plot, `p` and adds new data, based on real amplitude values, `data`.

`animate(p,angle,magnitude)` removes all the current data polar plot, `p` and adds new data sets of angle vectors and corresponding magnitude matrices.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`data` — Antenna or array data
real length-*M* vector | real *M*-by-*N* matrix | real *N-D* array | complex vector or matrix

Antenna or array data, specified as one of the following:

- A real length-*M* vector, where *M* contains the magnitude values with angles assumed to be $\frac{(0{:}M-1)}{M} \times 360^{\circ}$ degrees.
- A real *M*-by-*N* matrix, where *M* contains the magnitude values and *N* contains the independent data sets. Each column in the matrix has angles taken from the vector $\frac{(0{:}M-1)}{M} \times 360^{\circ}$ degrees. The set of each angle can vary for each column.
- A real *N-D* array, where *N* is the number of dimensions. Arrays with dimensions `2` and greater are independent data sets.
- A complex vector or matrix, where `data` contains Cartesian coordinates (*(x,y)* of each point. *x* contains the real part of `data` and *y* contains the imaginary part of `data`.

When data is in a logarithmic form such as dB, magnitude values can be negative. In this case,`polarpattern` plots the lowest magnitude values at the origin of the polar plot and highest magnitude values at the maximum radius.

`angle` — Set of angles
vector in degrees

Set of angles, specified as a vector in degrees.

`magnitude` — Set of magnitude values
vector | matrix

Set of magnitude values, specified as a vector or a matrix. For a matrix of magnitude values, each column is an independent set of magnitude values and corresponds to the same set of angles.

Examples

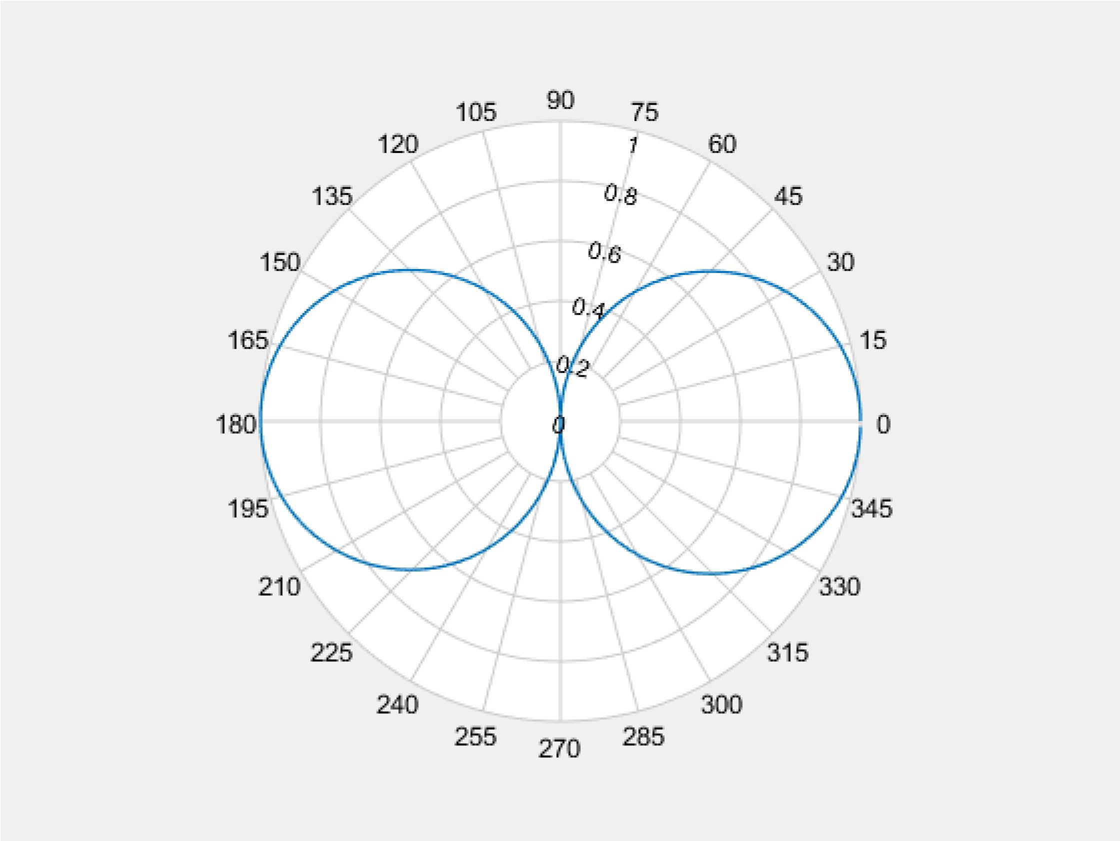
Animate Cosine Pattern Antenna Plot

Create a cosine-pattern antenna and plot the pattern from 0° to 360°.

```
az = [0:360];
p1 = abs(cosd(az));
```

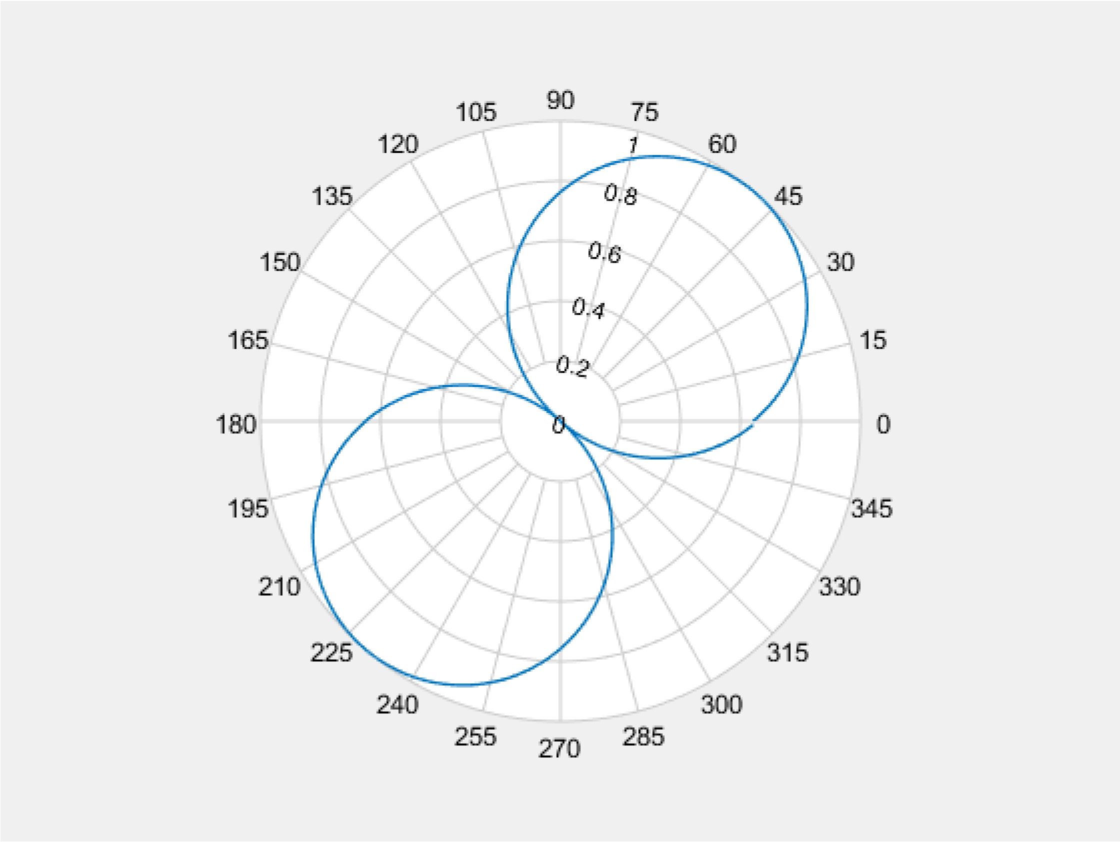
Plot the polar pattern.

```
P = polarpattern(p1);
```



Create a second cosine-pattern antenna rotated by 60°. Animate the pattern by adding this pattern.

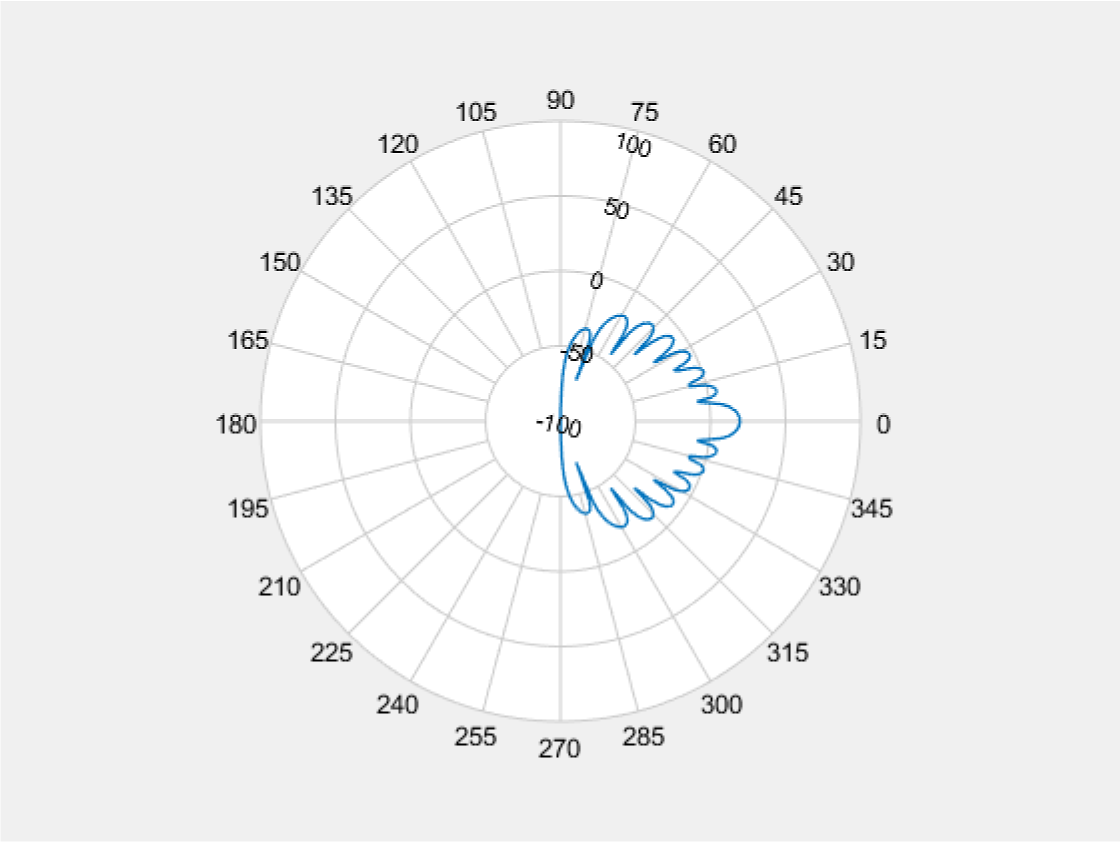
```
p2 = abs(cosd(az - 50));
animate(P,p2);
```



Animate Existing Polar Azimuth Plot Data

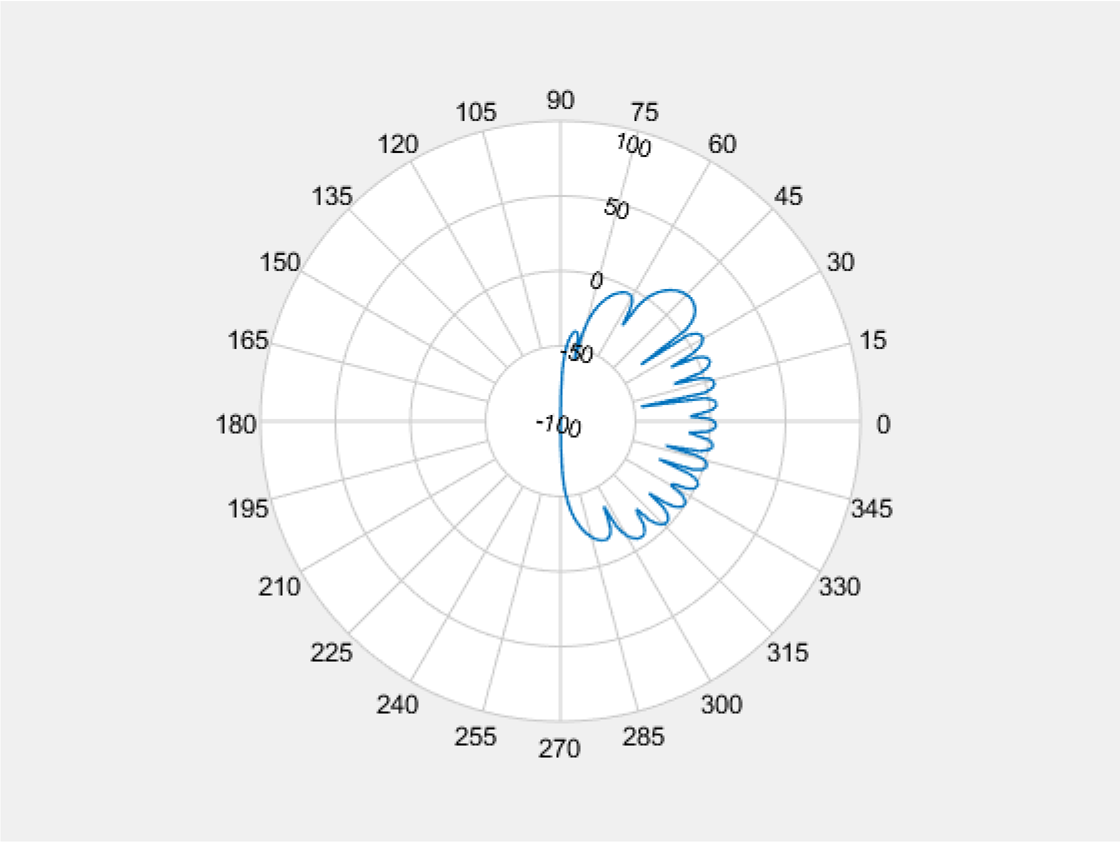
Create a 15-element ULA of cosine antennas with elements spaced one-half wavelength apart. Plot the directivity of the array at 20 GHz.

```
fc = 20.0e9;
c = physconst('Lightspeed');
lam = c/fc;
angs = [-180:1:180];
antenna = phased.CosineAntennaElement('FrequencyRange',[1.0e9,100.0e9],...
    'CosinePower',[2.5 2.5]);
array = phased.ULA('Element',antenna,'NumElements',15,'ElementSpacing',lam/2);
a = pattern(array,fc,angs,0);
P = polarpattern(angs,a);
```



Then, steer the array to 45° and, using the `animate` method, replace the existing polar plot with the steered array directivity.

```
steervec = phased.SteeringVector('SensorArray',array,'PropagationSpeed',c,...
    'IncludeElementResponse',true);
sv = steervec(fc,[45;0]);
a1 = pattern(array,fc,angs,0,'Weights',sv);
animate(P,angs,a1);
```



Version History
Introduced in R2016a

See Also
`add` | `addCursor` | `createLabels` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

createLabels

Create legend labels for polar plot

Syntax

`createLabels(p,format,array)`

Description

`createLabels(p,format,array)` adds the specified `format` label to each `array` of the polar plot `p`. The labels are stored as a cell array in the `LegendLabels` property of `p`.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`format` — Format for legend label
cell array

Format for legend label added to the polar plot, specified as a cell array. For more information on legend label format see, `legend`.

Data Types: `char`

`array` — Values to apply to `format`
array

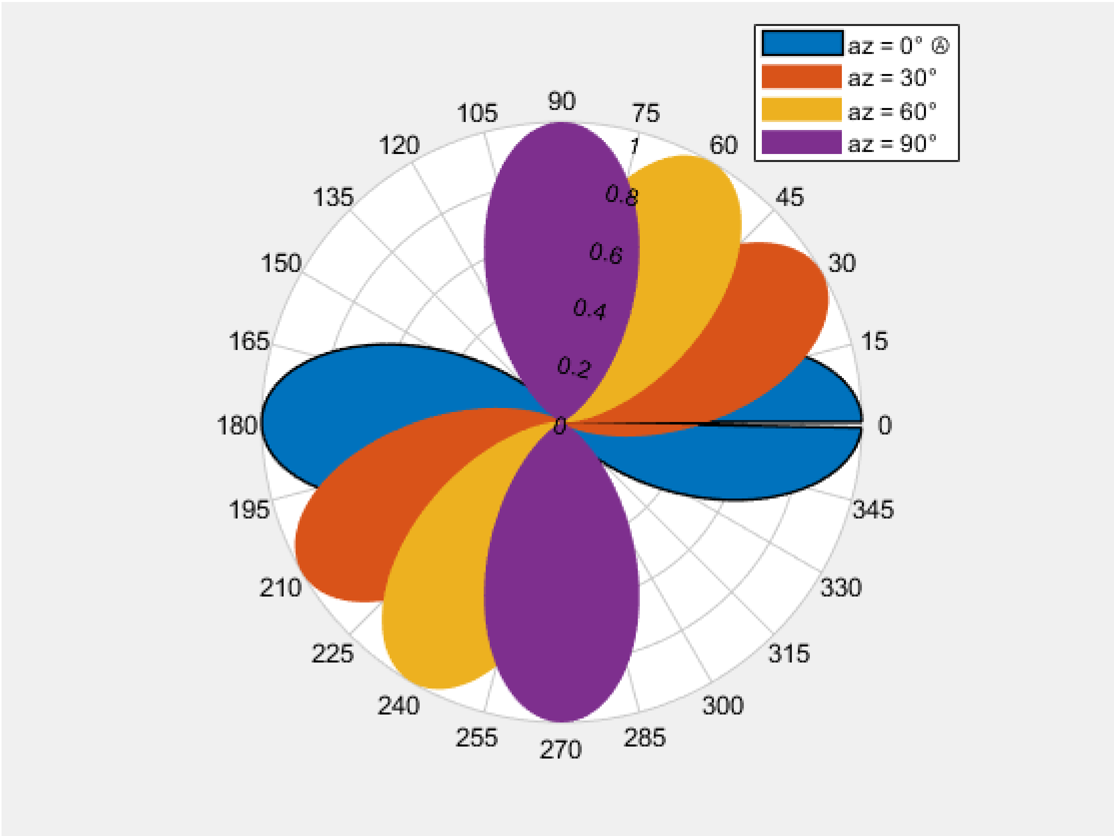
Values to apply to `format` , specified as an array. The values can be an array of angles or array of magnitude.

Examples

Add Legend Label to Polar Plot

Create a polar plot of cosine powers rotated in 30° increments. Generate a legend label for this plot.

```
az = [0:359]';
a1 = abs(cosd(az).^5);
a2 = abs(cosd(az - 30).^5);
a3 = abs(cosd(az - 60).^5);
a4 = abs(cosd(az - 90).^5);
P = polarpattern([a1,a2,a3,a4],'Style','filled');
createLabels(P,'az = %d#deg',0:30:90)
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `findLobes` | `replace` | `showPeaksTable` | `showSpan`

findLobes

Main, back, and side lobe data

Syntax

```
L = findLobes(p)
L = findLobes(p,index)
```

Description

`L = findLobes(p)` returns a structure, `L`, defining the main, back, and side lobes of the antenna or array radiation pattern in the specified polar plot, `p`.

`L = findLobes(p,index)` returns the radiation pattern lobes from the data set specified in `index`.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

index — Index of data set
scalar

Index of data set, specified as a scalar.

Output Arguments

L — Main,back, and side lobe data
structure

Main,back, and side lobe data, returned as a structure.

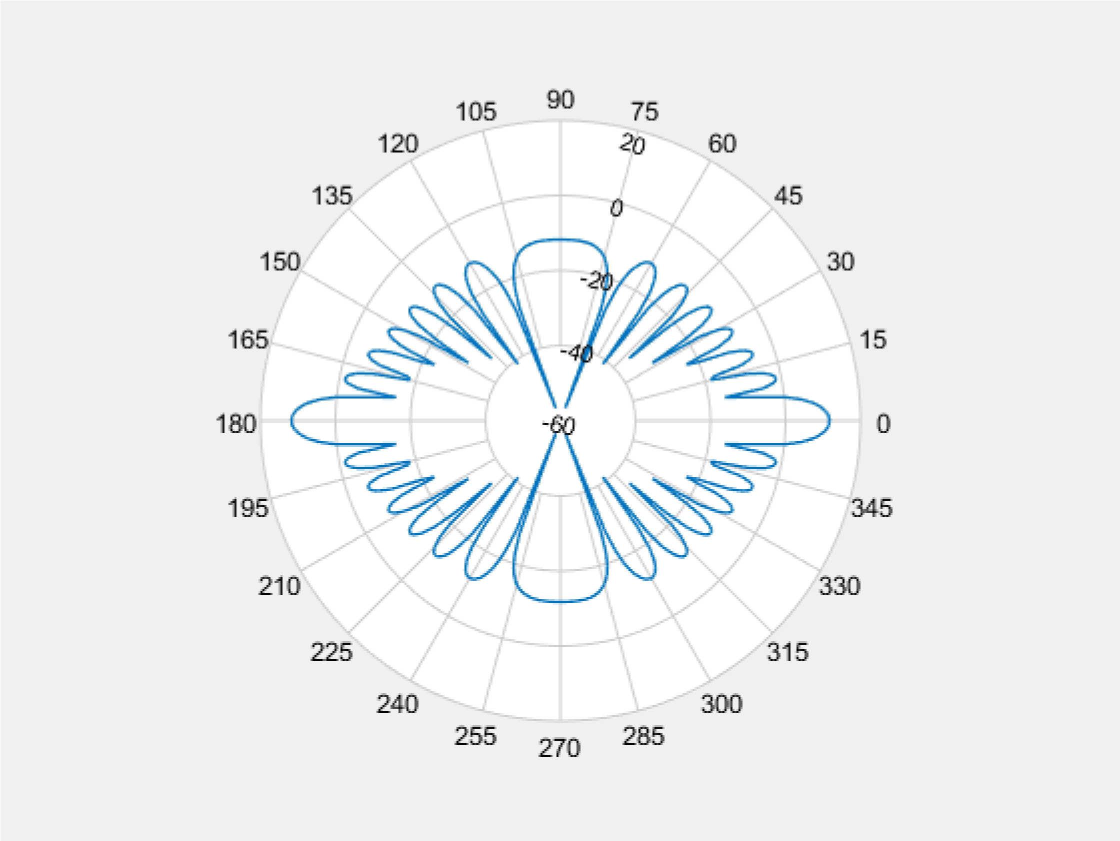
Examples

Find Lobes of Isotropic Antenna ULA

Create a 15-element ULA of isotropic antenna with elements spaced one-half wavelength apart. Plot the directivity of the array at 20 GHz. Then, find the mainlobe, sidelobe, and backlobe directions of the array pattern.

```
fc = 20.0e9;
c = physconst('Lightspeed');
lam = c/fc;
angs = [-180:1:180];
antenna = phased.IsotropicAntennaElement('FrequencyRange',[1.0e9,100.0e9]);
array = phased.ULA('Element',antenna,'NumElements',15,'ElementSpacing',lam/2);
```

```
a = pattern(array,fc,angs,0);
P = polarpattern(angs,a);
```



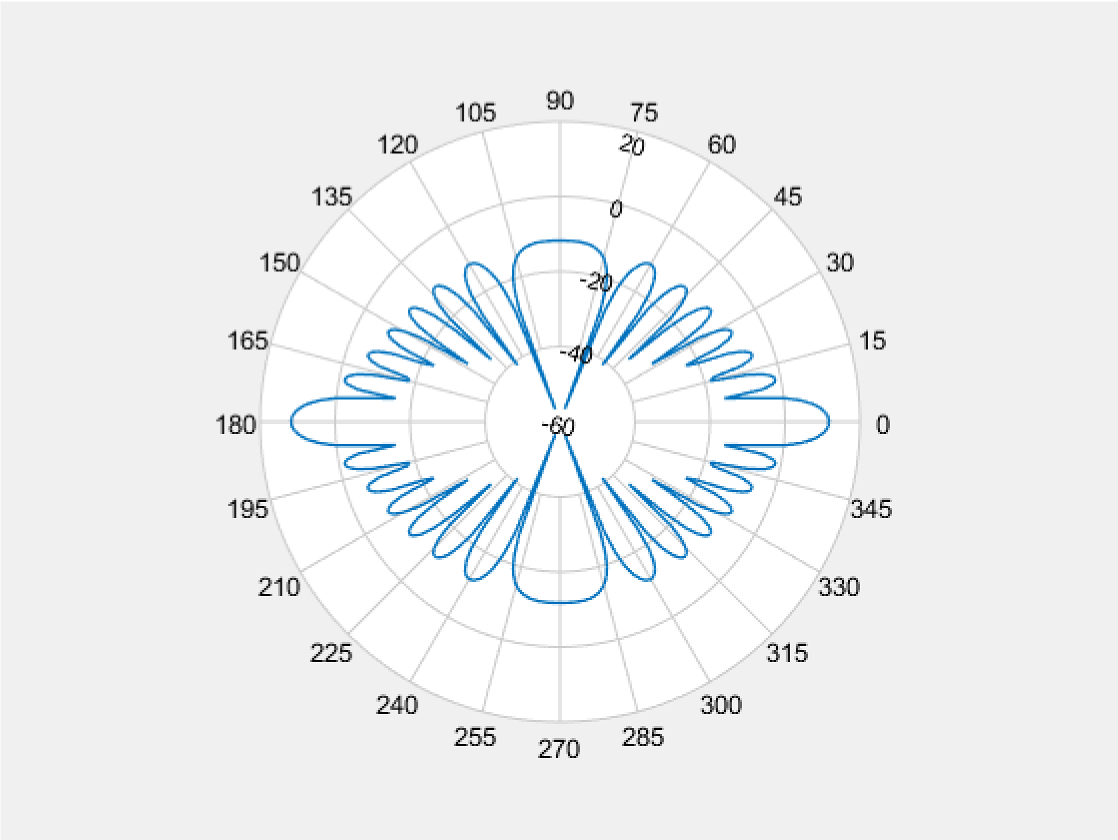

```
L = findLobes(P)
```

```
L = struct with fields:
      mainLobe: [1x1 struct]
      backLobe: [1x1 struct]
     sideLobes: [1x1 struct]
            FB: 0
           SLL: 0
          HPBW: 8.0000
          FNBW: 16.0000
         FBIdx: [181 1]
        SLLIdx: [181 361]
       HPBWIdx: [357 5]
       HPBWAng: [176 -176]
       FNBWIdx: [173 189]
```

Find Lobes of Steered Isotropic Antenna ULA Patterns

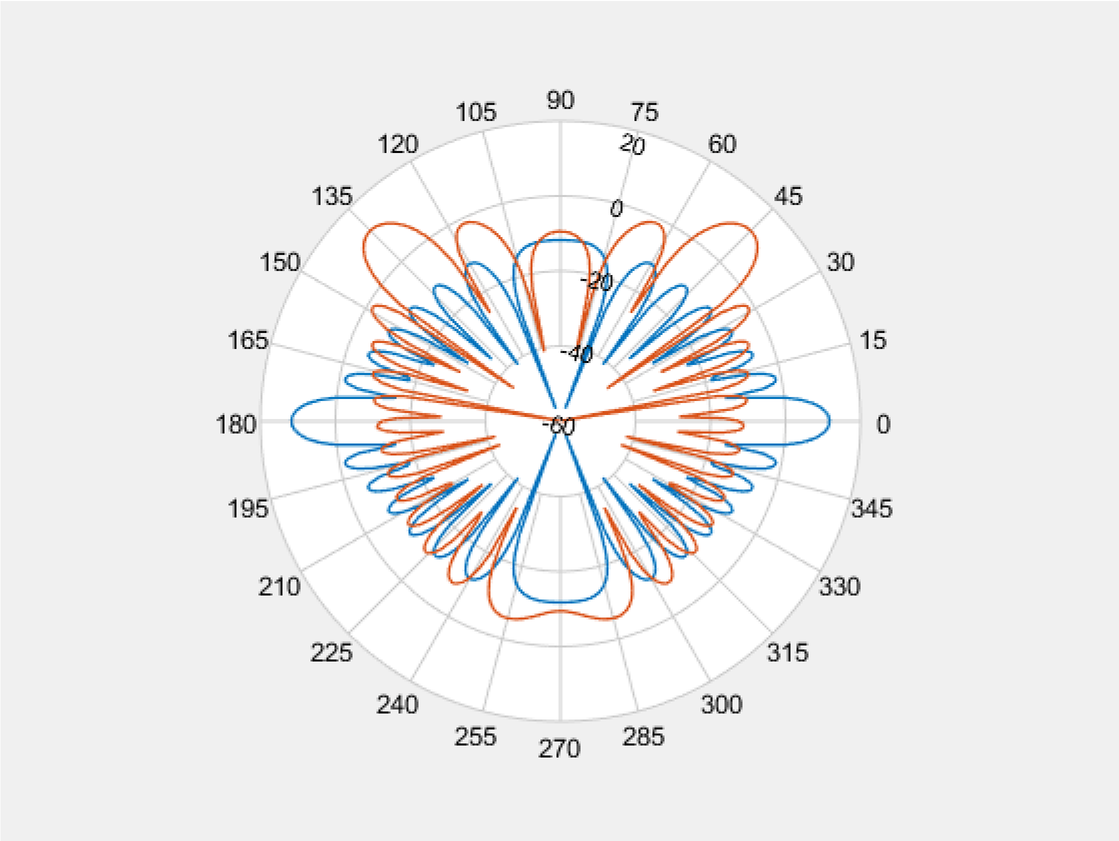
Create a 15-element ULA of isotropic antenna with elements spaced one-half wavelength apart. Plot the directivity of the array at 20 GHz. Then steer the array to 45° azimuth and plot the directivity. Then, find the mainlobe, sidelobe, and backlobe directions of the array pattern.

```
fc = 20.0e9;
c = physconst('Lightspeed');
lam = c/fc;
angs = [-180:1:180];
antenna = phased.IsotropicAntennaElement('FrequencyRange',[1.0e9,100.0e9]);
array = phased.ULA('Element',antenna,'NumElements',15,'ElementSpacing',lam/2);
a = pattern(array,fc,angs,0);
P = polarpattern(angs,a);
```



Steer the array to 45° azimuth and add the steered pattern to the polar plot.

```
steervec = phased.SteeringVector('SensorArray',array,'PropagationSpeed',c);
sv = steervec(fc,[45;0]);
a1 = pattern(array,fc,angs,0,'Weights',sv);
add(P,angs,a1);
```



Find the lobes of the steered pattern.

```
L = findLobes(P,2);
L.mainLobe
```

```
ans = struct with fields:
        index: 226
    magnitude: 11.7609
        angle: 45
       extent: [216 238]
```

Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `createLabels` | `replace` | `showPeaksTable` | `showSpan`

replace

Replace polar plot data with new data

Syntax

```
replace(p,data)
replace(p,angle,magnitude)
```

Description

`replace(p,data)` removes all data from polar plot, `p` and adds new data based on real amplitude values, `data`.

`replace(p,angle,magnitude)` removes all the current data and adds new data sets of angle vectors and corresponding magnitude matrices to the polar plot, `p`.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`data` — Antenna or array data
real length-*M* vector | real *M*-by-*N* matrix | real *N-D* array | complex vector or matrix

Antenna or array data, specified as one of the following:

- A real length-*M* vector, where *M* contains the magnitude values with angles assumed to be $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees.
- A real *M*-by-*N* matrix, where *M* contains the magnitude values and *N* contains the independent data sets. Each column in the matrix has angles taken from the vector $\frac{(0:M-1)}{M} \times 360^{\circ}$ degrees. The set of each angle can vary for each column.
- A real *N-D* array, where *N* is the number of dimensions. Arrays with dimensions `2` and greater are independent data sets.
- A complex vector or matrix, where `data` contains Cartesian coordinates (*(x,y)* of each point. *x* contains the real part of `data` and *y* contains the imaginary part of `data`.

When data is in a logarithmic form such as dB, magnitude values can be negative. In this case,`polarpattern` plots the lowest magnitude values at the origin of the polar plot and highest magnitude values at the maximum radius.

`angle` — Set of angles
vector in degrees

Set of angles, specified as a vector in degrees.

magnitude — Set of magnitude values
vector | matrix

Set of magnitude values, specified as a vector or a matrix. For a matrix of magnitude values, each column is an independent set of magnitude values and corresponds to the same set of angles.

Examples

Replace Cosine Polar Plot with Rotated Cosine Polar Plot

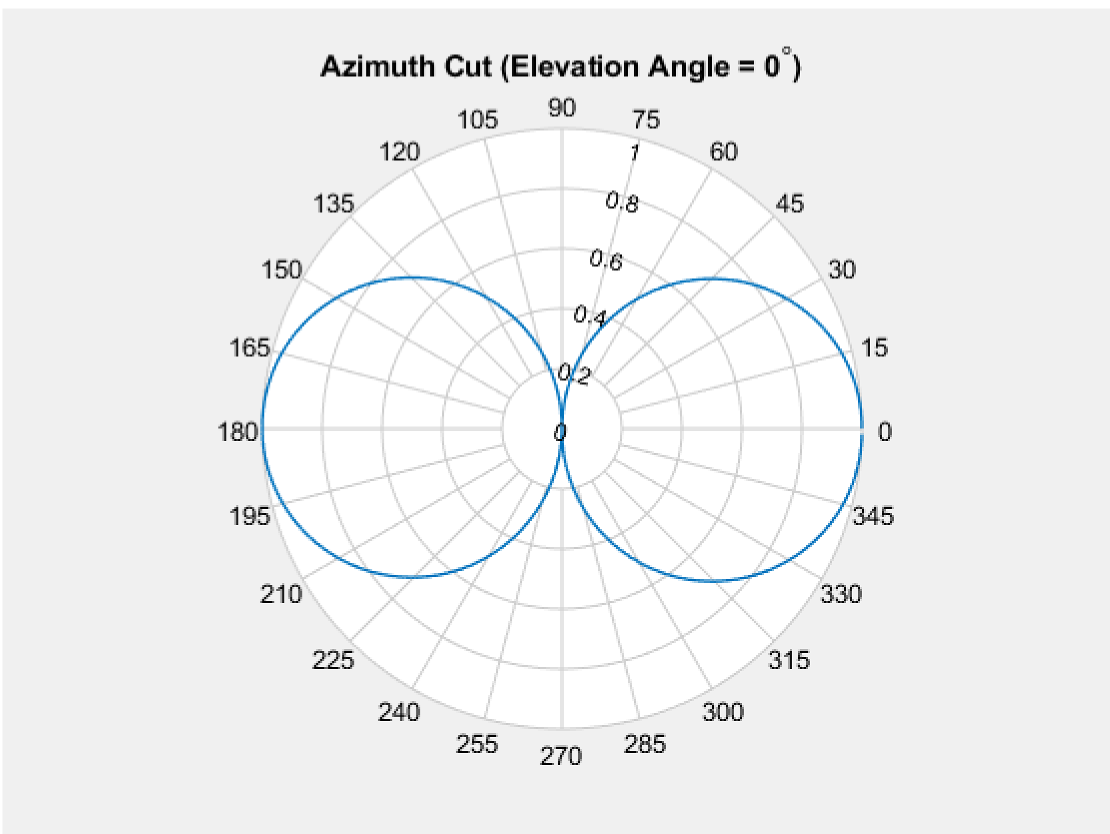
Plot cosine pattern in polar coordinates. Specify a cosine antenna pattern from 0° to 360° in azimuth at 0° elevation. Then, plot the antenna pattern using `polarpattern`.

Create the pattern.

```
az = [0:360];
a = abs(cosd(az));
```

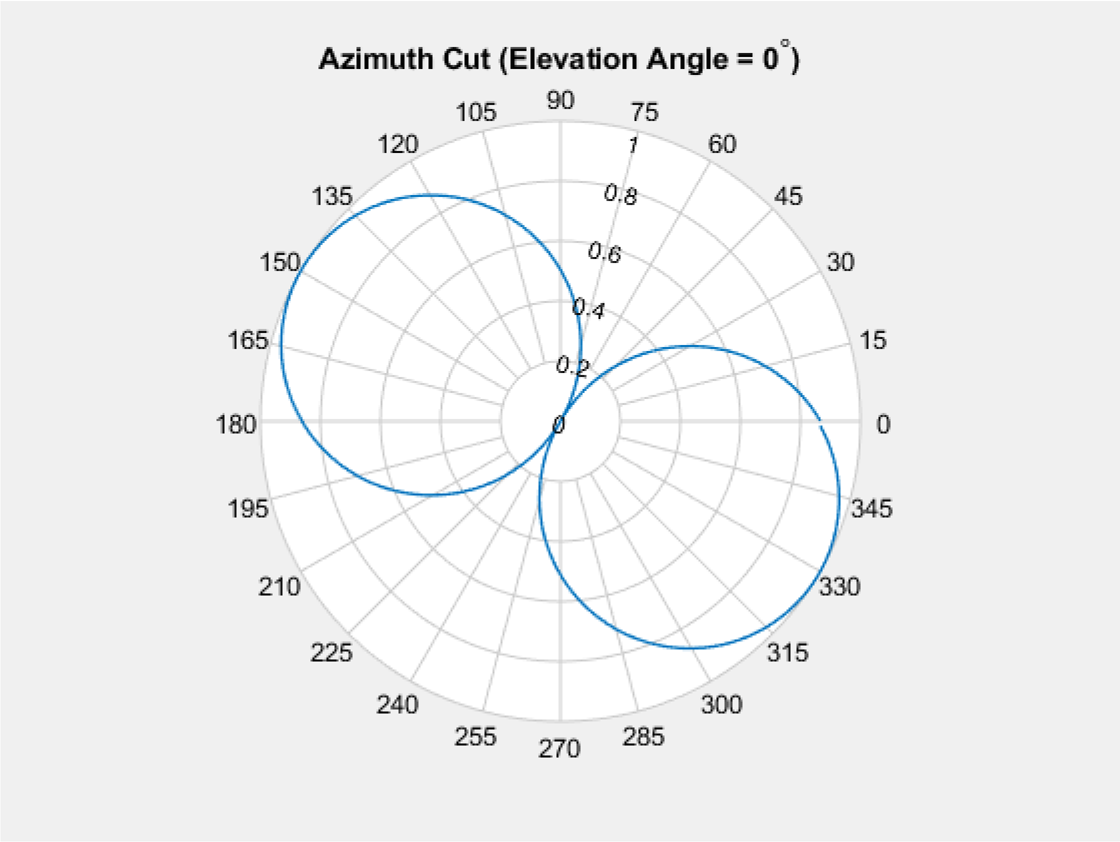
Plot the polar pattern of the antenna for an azimuth cut at 0° elevation.

```
P = polarpattern(a,'TitleTopTextInterpreter','tex','TitleTop','Azimuth Cut (Elevation Angle = 0^
```



Replace this plot with a rotated cosine pattern.

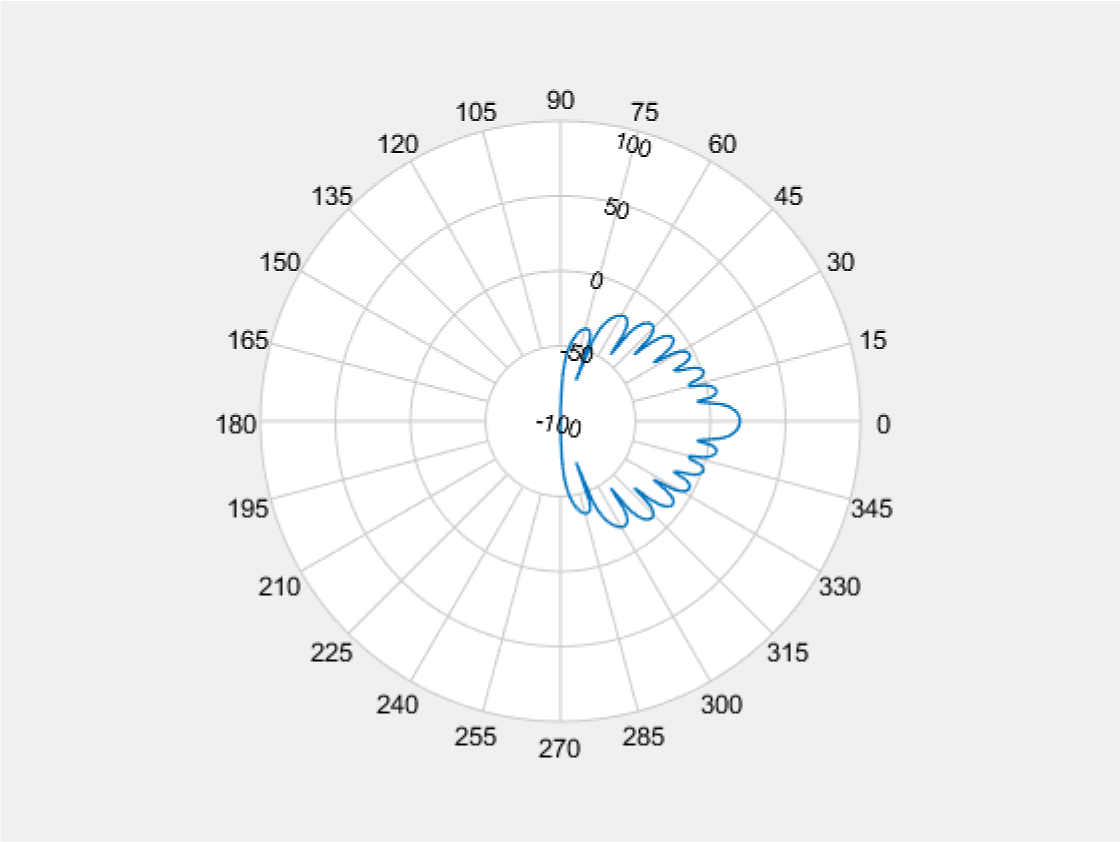
```
a = abs(cosd(az + 30.0));
replace(P,a);
```



Replace Polar Plot Data with New Angle-Magnitude Data

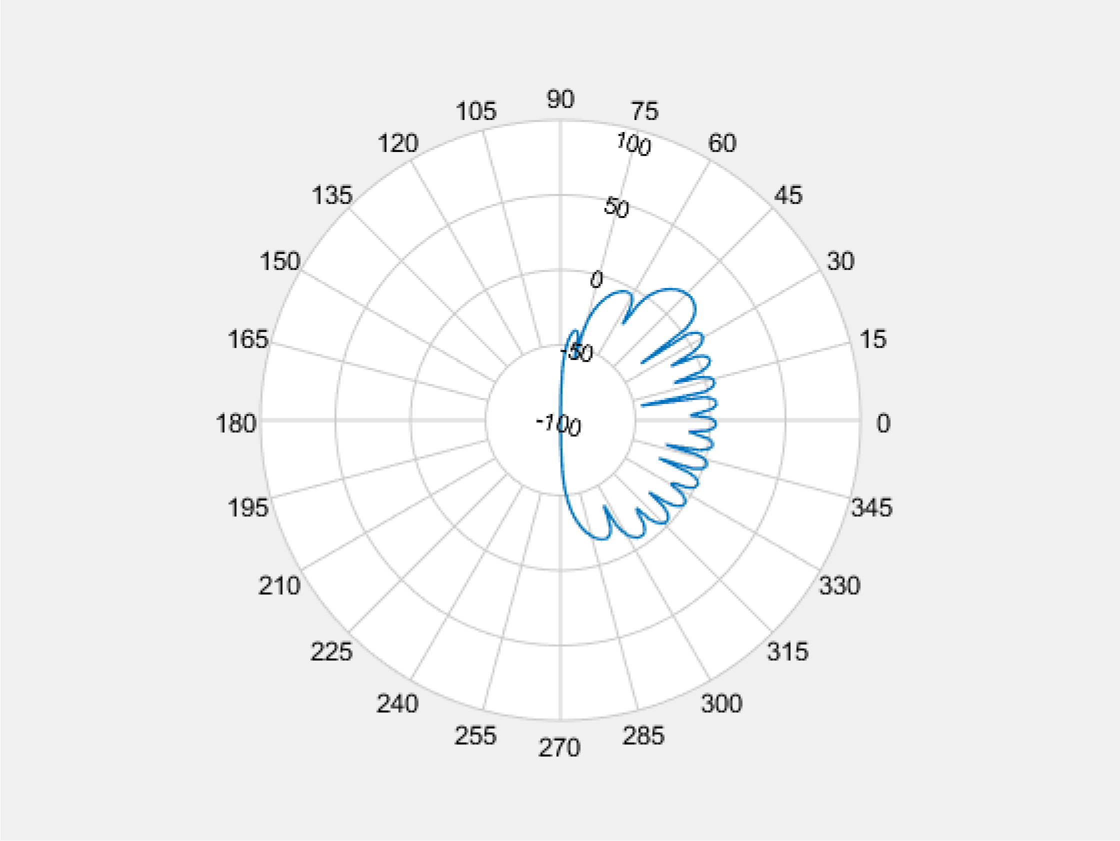
Create a 15-element ULA of cosine antennas with elements spaced one-half wavelength apart. Plot the directivity of the array at 20 GHz.

```
fc = 20.0e9;
c = physconst('Lightspeed');
lam = c/fc;
angs = [-180:1:180];
antenna = phased.CosineAntennaElement('FrequencyRange',[1.0e9,100.0e9],...
    'CosinePower',[2.5 2.5]);
array = phased.ULA('Element',antenna,'NumElements',15,'ElementSpacing',lam/2);
a = pattern(array,fc,angs,0);
P = polarpattern(angs,a);
```



Then, steer the array to 45° and, using the `replace` method, replace the existing polar plot with the steered array directivity.

```
steervec = phased.SteeringVector('SensorArray',array,'PropagationSpeed',c,...
    'IncludeElementResponse',true);
sv = steervec(fc,[45;0]);
a1 = pattern(array,fc,angs,0,'Weights',sv);
replace(P,angs,a1);
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `createLabels` | `findLobes` | `showPeaksTable` | `showSpan`

showPeaksTable

Show or hide peak marker table

Syntax

```
showPeaksTable(p,vis)
```

Description

`showPeaksTable(p,vis)` shows or hides a table of the peak values. By default, the peak values table is visible.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

vis — Show or hide peaks table
0 | 1

Show or hide peaks table, specified as `0` or `1`.

Examples

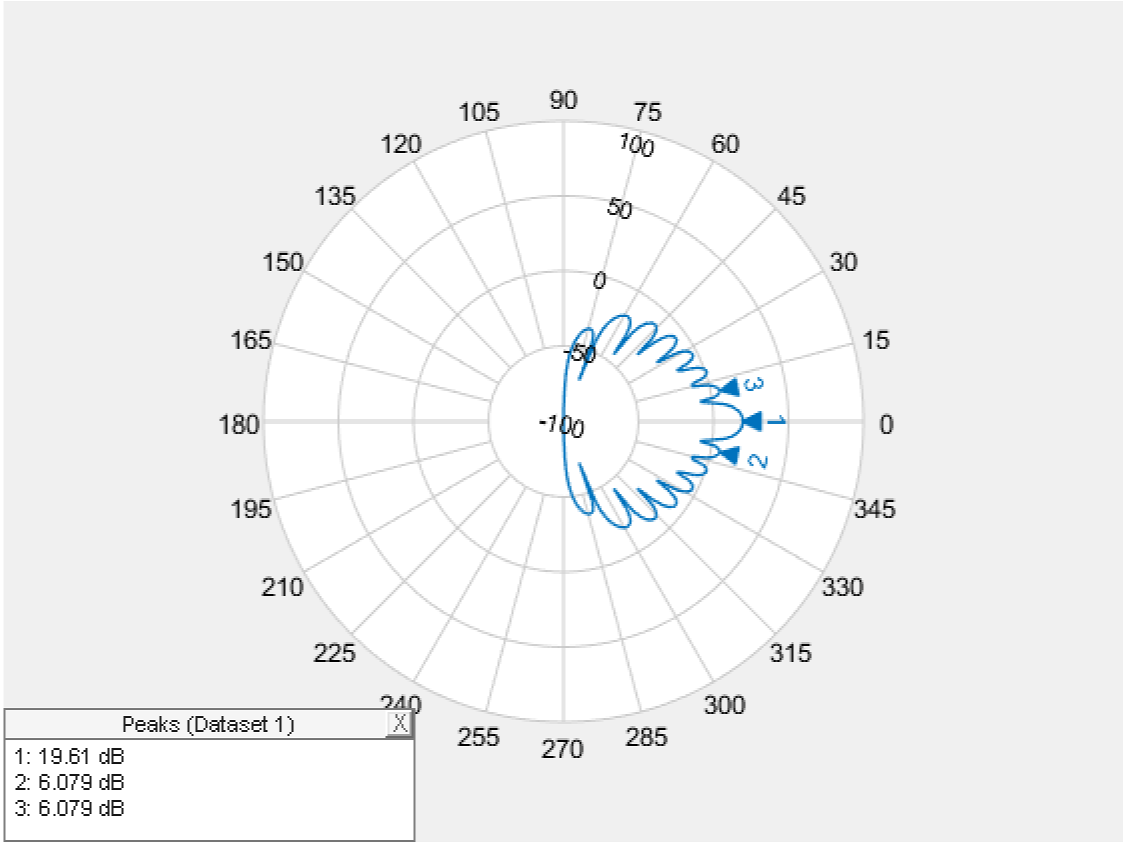
Peaks of ULA Array in Polar Pattern

Create a 15-element ULA of cosine antennas with elements spaced one-half wavelength apart. Then, plot the directivity of the array at 20 GHz.

```
fc = 20.0e9;
c = physconst('Lightspeed');
lam = c/fc;
angs = [-180:1:180];
antenna = phased.CosineAntennaElement('FrequencyRange',[1.0e9,100.0e9],...
    'CosinePower',[2.5 2.5]);
array = phased.ULA('Element',antenna,'NumElements',15,'ElementSpacing',lam/2);
```

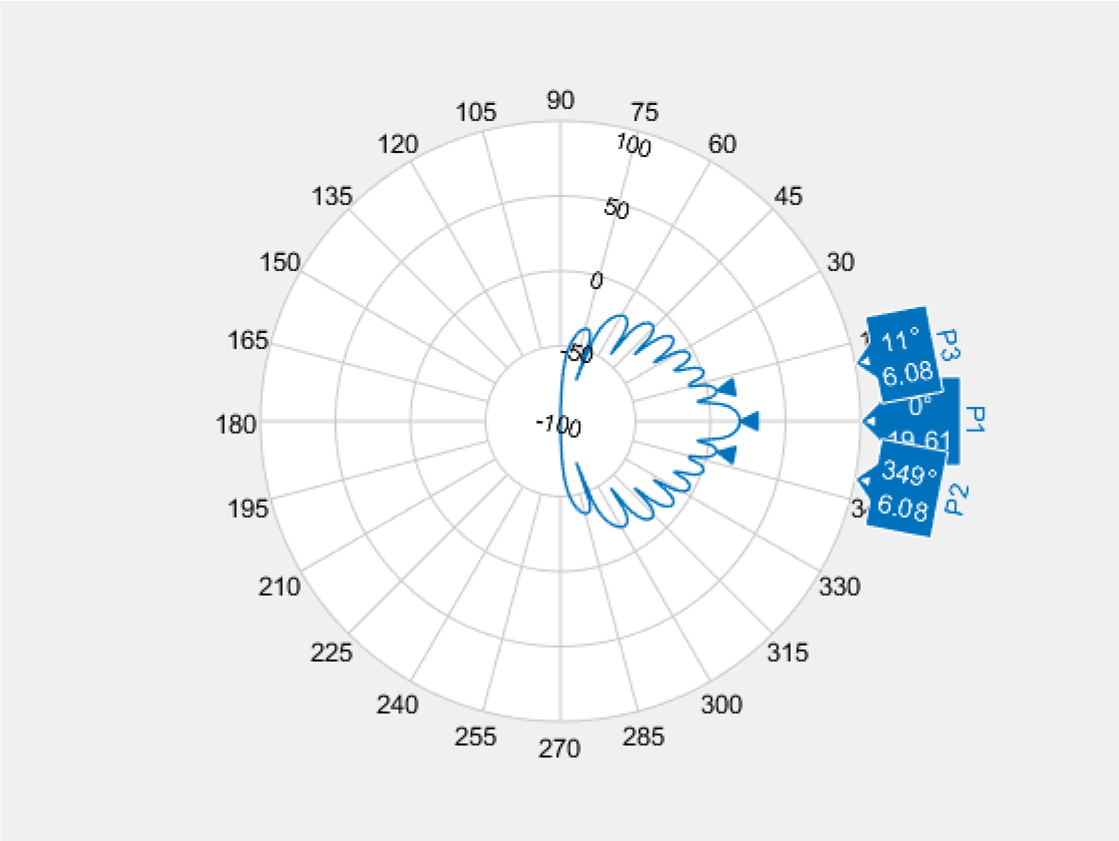
Plot the polar pattern and show three peaks of the antenna. When creating a `polarpattern` plot, if you specify the Peaks property, the peaks table is displayed by default.

```
a = pattern(array,fc,angs,0);
P = polarpattern(angs,a,'Peaks',3);
```



Hide the table. When the peaks table is hidden, the peak markers display the peak values.

```
showPeaksTable(P,0);
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `createLabels` | `findLobes` | `replace` | `showSpan`

PolarPattern Properties

Control appearance and behavior of polar plot

Description

Polar pattern properties control the appearance and behavior of the polar pattern object. By changing property values, you can modify certain aspects of the polar plot. To change the default properties use:

```
p = polarpattern(____,Name,Value)
```

To view all the properties of the polar pattern object use:

```
details(p)
```

Properties

Antenna Metrics

`'AntennaMetrics'` — Show antenna metric
`0` (default) | `1`

Show antenna metrics, specified as a comma-separated pair consisting of `'AntennaMetrics'` and `0` or `1`. Antenna metric displays main, back, and side lobes of antenna/array pattern passed as input.

Data Types: `logical`

`'Peaks'` — Maximum number of peaks to compute for each data set
positive integer | vector of integers

Maximum number of peaks to compute for each data set, specified as a comma-separated pair consisting of `'Peaks'` and a positive scalar or vector of integers.

Data Types: `double`

Angle Properties

`'AngleAtTop'` — Angle at top of polar plot
`90` (default) | scalar in degrees

Angle at the top of the polar plot, specified as a comma-separated pair consisting of `'AngleAtTop'` and a scalar in degrees.

Data Types: `double`

`'AngleLim'` — Visible polar angle span
`[0 360]` (default) | 1-by-2 vector of real values

Visible polar angle span, specified as a comma-separated pair consisting of `'AngleLim'` and a 1-by-2 vector of real values.

Data Types: `double`

'AngleLimVisible' — Show interactive angle limit cursors
0 (default) | 1

Show interactive angle limit cursors, specified as a comma-separated pair consisting of 'AngleLimVisible' and 0 or 1.

Data Types: logical

'AngleDirection' — Direction of increasing angle
'ccw' (default) | 'cw'

Direction of increasing angle, specified as a comma-separated pair consisting of 'AngleDirection' and 'ccw' (counterclockwise) or 'cw' (clockwise).

Data Types: char

'AngleResolution' — Number of degrees between radial lines
15 (default) | scalar in degrees

Number of degrees between radial lines depicting angles in the polar plot, specified as a comma-separated pair consisting of 'AngleResolution' and a scalar in degrees.

Data Types: double

'AngleTickLabelRotation' — Rotate angle tick labels
0 (default) | 1

Rotate angle tick labels, specified as a comma-separated pair consisting of 'AngleTickLabelRotation' and 0 or 1.

Data Types: logical

'AngleTickLabelVisible' — Show angle tick labels
1 (default) | 0

Show angle tick labels, specified as a comma-separated pair consisting of 'AngleTickLabelVisible' and 0 or 1.

Data Types: logical

'AngleTickLabelFormat' — Format for angle tick labels
360 (default) | 180

Format for angle tick labels, specified as a comma-separated pair consisting of 'AngleTickLabelFormat' and 360 degrees or 180 degrees.

Data Types: double

'AngleFontSizeMultiplier' — Scale factor of angle tick font
1 (default) | numeric value greater than zero

Scale factor of angle tick font, specified as a comma-separated pair consisting of 'AngleFontSizeMultiplier' and a numeric value greater than zero.

Data Types: double

'Span' — Show angle span measurement
0 (default) | 1

Show angle span measurement, specified as a comma-separated pair consisting of `'Span'` and `0` or `1`.

Data Types: `logical`

`'ZeroAngleLine'` — Highlight radial line at zero degrees
`0` (default) | `1`

Highlight radial line at zero degrees, specified as a comma-separated pair consisting of `'ZeroAngleLine'` and `0` or `1`.

Data Types: `logical`

`'DisconnectAngleGaps'` — Show gaps in line plots with nonuniform angle spacing
`1` (default) | `0`

Show gaps in line plots with nonuniform angle spacing, specified as a comma-separated pair consisting of `'DisconnectAngleGaps'` and `0` or `1`.

Data Types: `logical`

Magnitude Properties

`'MagnitudeAxisAngle'` — Angle of magnitude tick label radial line
`75` (default) | real scalar in degrees

Angle of magnitude tick label radial line, specified as a comma-separated pair consisting of `'MagnitudeAxisAngle'` and real scalar in degrees.

Data Types: `double`

`'MagnitudeTick'` — Magnitude ticks
`[0 0.2 0.4 0.6 0.8]` (default) | `1`-by-`N` vector

Magnitude ticks, specified as a comma-separated pair consisting of `'MagnitudeTick'` and a `1`-by-`N` vector, where `N` is the number of magnitude ticks.

Data Types: `double`

`'MagnitudeTickLabelVisible'` — Show magnitude tick labels
`1` (default) | `0`

Show magnitude tick labels, specified as a comma-separated pair consisting of `'MagnitudeTickLabelVisible'` and `0` or `1`.

Data Types: `logical`

`'MagnitudeLim'` — Minimum and maximum magnitude limits
`[0 1]` (default) | two-element vector of real values

Minimum and maximum magnitude limits, specified as a comma-separated pair consisting of `'MagnitudeLim'` and a two-element vector of real values.

Data Types: `double`

`'MagnitudeLimMode'` — Determine magnitude dynamic range
`'auto'` (default) | `'manual'`

Determine magnitude dynamic range, specified as a comma-separated pair consisting of `'MagnitudeLimMode'` and `'auto'` or `'manual'`.

Data Types: `char`

`'MagnitudeAxisAngleMode'` — Determine angle for magnitude tick labels
`'auto'` (default) | `'manual'`

Determine angle for magnitude tick labels, specified as a comma-separated pair consisting of `'MagnitudeAxisAngleMode'` and `'auto'` or `'manual'`.

Data Types: `char`

`'MagnitudeTickMode'` — Determine magnitude tick locations
`'auto'` (default) | `'manual'`

Determine magnitude tick locations, specified as a comma-separated pair consisting of `'MagnitudeTickMode'` and `'auto'` or `'manual'`.

Data Types: `char`

`'MagnitudeUnits'` — Magnitude units
`'dB'` | `'dBLoss'`

Magnitude units, specified as a comma-separated pair consisting of `'MagnitudeUnits'` and `'db'` or `'dBLoss'`.

Data Types: `char`

`'MagnitudeFontSizeMultiplier'` — Scale factor of magnitude tick font
`0.9000` (default) | numeric value greater than zero

Scale factor of magnitude tick font, specified as a comma-separated pair consisting of `'MagnitudeFontSizeMultiplier'` and a numeric value greater than zero.

Data Types: `double`

Miscellaneous Properties

`'View'` — View section of Smith plot
`'full'` (default) | `'top'` | `'bottom'` | `'left'` | `'right'` | `'top-left'` | `'top-right'` | `'bottom-left'` | `'bottom-right'` | string scalar | character vector

View section of Smith plot, specified as a string scalar or character vector. Smith plot can be viewed by setting `View` property to one of property values in this table.

View Property Effect

Property Value	Effect
'full'	Full Smith plot is viewed.
'top'	Top-half of the Smith plot is viewed.
'bottom'	Bottom-half of the Smith plot is viewed.
'left'	Left-half of the Smith plot is viewed.
'right'	Right-half of the Smith plot is viewed.
'top-left'	Top-left of the Smith plot is viewed.
'top-right'	Top-right of the Smith plot is viewed.
'bottom-left'	Bottom-left of the Smith plot is viewed.
'bottom-right'	Bottom-right of the Smith plot is viewed.

Data Types: `char` | `string`

'NormalizeData' — Normalize each data trace to maximum value

`0` (default) | `1`

Normalize each data trace to maximum value, specified as a comma-separated pair consisting of `'NormalizeData'` and `0` or `1`.

Data Types: `logical`

'ConnectEndpoints' — Connect first and last angles

`0` (default) | `1`

Connect first and last angles, specified as a comma-separated pair consisting of `'ConnectEndpoints'` and `0` or `1`.

Data Types: `logical`

'Style' — Style of polar plot display

`'line'` (default) | `'filled'`

Style of polar plot display, specified as a comma-separated pair consisting of `'Style'` and `'line'` or `'filled'`.

Data Types: `char`

'TemporaryCursor' — Create temporary cursor

`0` (default) | `1`

Create a temporary cursor, specified as a comma-separated pair consisting of `'TemporaryCursor'` and `0` or `1`.

Data Types: `logical`

'ToolTips' — Show tool tips

`1` (default) | `0`

Show tool tips when you hover over a polar plot element, specified as a comma-separated pair consisting of `'ToolTips'` and `0` or `1`.

Data Types: `logical`

'ClipData' — Clip data to outer circle
0 (default) | 1

Clip data to outer circle, specified as a comma-separated pair consisting of 'ClipData' and 0 or 1.

Data Types: logical

'CleanData' — Cleans data
0 (default) | 1

Cleans data removing any Inf or NaN from the data. The property is specified as a comma-separated pair consisting of 'CleanData' and 0 or 1.

Data Types: logical

'NextPlot' — Directive on how to add next plot
'replace' (default) | 'new' | 'add'

Directive on how to add next plot, specified as a comma-separated pair consisting of 'NextPlot' and one of the values in the table:

Property Value	Effect
'new'	Creates a figure and uses it as the current figure.
'add'	Adds new graphics objects without clearing or resetting the current figure.
'replace'	Removes all axes objects and resets figure properties to their defaults before adding new graphics objects.

Legend and Title Properties

'LegendLabels' — Data tables for legend annotation
character vector | cell array of character vectors

Data tables for legend annotation, specified as a comma-separated pair consisting of 'LegendLabels' and a character vector or cell array of character vectors. Ⓐ denotes the active line for interactive operation.

Data Types: char

'LegendVisible' — Show legend label
0 (default) | 1

Show legend label, specified as a comma-separated pair consisting of 'LegendVisible' and 0 or 1.

Data Types: logical

'TitleTop' — Title to display above the polar plot
character vector

Title to display above the polar plot, specified as a comma-separated pair consisting of 'TitleTop' and a character vector.

Data Types: char

'TitleBottom' — Title to display below the polar plot
character vector

Title to display below the polar plot, specified as a comma-separated pair consisting of 'TitleBottom' and a character vector.

Data Types: char

'TitleTopOffset' — Offset between top title and angle ticks
0.1500 (default) | scalar

Offset between top title and angle ticks, specified as a comma-separated pair consisting of 'TitleTopOffset' and a scalar. The value must be in the range [-0.5,0.5].

Data Types: double

'TitleBottomOffset' — Offset between bottom title and angle ticks
0.1500 (default) | scalar

Offset between bottom title and angle ticks, specified as a comma-separated pair consisting of 'TitleBottomOffset' and a scalar. The value must be in the range [-0.5,0.5].

Data Types: double

'TitleTopFontSizeMultiplier' — Scale factor of top title font
1.1000 (default) | numeric value greater than zero

Scale factor of top title font, specified as a comma-separated pair consisting of 'TitleTopFontSizeMultiplier' and a numeric value greater than zero.

Data Types: double

'TitleBottomFontSizeMultiplier' — Scale factor of bottom title font
0.9000 (default) | numeric value greater than zero

Scale factor of bottom title font, specified as a comma-separated pair consisting of 'TitleBottomFontSizeMultiplier' and a numeric value greater than zero.

Data Types: double

'TitleTopFontWeight' — Thickness of top title font
'bold' (default) | 'normal'

Thickness of top title font, specified as a comma-separated pair consisting of 'TitleTopFontWeight' and 'bold' or 'normal.

Data Types: char

'TitleBottomFontWeight' — Thickness of bottom title font
'normal' (default) | 'bold'

Thickness of bottom title font, specified as a comma-separated pair consisting of 'TitleBottomFontWeight' and 'bold' or 'normal.

Data Types: char

'TitleTopTextInterpreter' — Interpretation of top title characters
'none' (default) | 'tex' | 'latex'

Interpretation of top title characters, specified as a comma-separated pair consisting of `'TitleTopTextInterpreter'` and:

- `'tex'` — Interpret using a subset of TeX markup
- `'latex'` — Interpret using LaTeX markup
- `'none'` — Display literal characters

TeX Markup

By default, MATLAB supports a subset of TeX markup. Use TeX markup to add superscripts and subscripts, modify the text type and color, and include special characters in the text.

This table lists the supported modifiers when the `TickLabelInterpreter` property is set to `'tex'`, which is the default value. Modifiers remain in effect until the end of the text, except for superscripts and subscripts which only modify the next character or text within curly braces `{}`.

Modifier	Description	Example
`^{ }`	Superscript	`'text^{superscript}'`
`_{ }`	Subscript	`'text_{subscript}'`
`\bf`	Bold font	`'\bf text'`
`\it`	Italic font	`'\it text'`
`\sl`	Oblique font (rarely available)	`'\sl text'`
`\rm`	Normal font	`'\rm text'`
`\fontname{specifier}`	Set `specifier` as the name of a font family to change the font style. You can use this modifier with other modifiers.	`'\fontname{Courier} text'`
`\fontsize{specifier}`	Set `specifier` as a scalar numeric value to change the font size.	`'\fontsize{15} text'`
`\color{specifier}`	Set `specifier` as one of these colors: `red`, `green`, `yellow`, `magenta`, `blue`, `black`, `white`, `gray`, `darkGreen`, `orange`, or `lightBlue`.	`'\color{magenta} text'`
`\color[rgb]{specifier}`	Set `specifier` as a three-element RGB triplet to change the font color.	`'\color[rgb]{0,0.5,0.5} text'`

LaTeX Markup

To use LaTeX markup, set the `TickLabelInterpreter` property to `'latex'`. The displayed text uses the default LaTeX font style. The `FontName`, `FontWeight`, and `FontAngle` properties do not have an effect. To change the font style, use LaTeX markup within the text.

The maximum size of the text that you can use with the LaTeX interpreter is 1200 characters. For multiline text, the maximum size reduces by about 10 characters per line.

Data Types: `char`

'TitleBottomTextInterpreter' — Interpretation of bottom title characters

'none' (default) | 'tex' | 'latex'

Interpretation of bottom title characters, specified as a comma-separated pair consisting of 'TitleBottomTextInterpreter' and:

- 'tex' — Interpret using a subset of TeX markup
- 'latex' — Interpret using LaTeX markup
- 'none' — Display literal characters

TeX Markup

By default, MATLAB supports a subset of TeX markup. Use TeX markup to add superscripts and subscripts, modify the text type and color, and include special characters in the text.

This table lists the supported modifiers when the TickLabelInterpreter property is set to 'tex', which is the default value. Modifiers remain in effect until the end of the text, except for superscripts and subscripts which only modify the next character or the text within the curly braces {}.

Modifier	Description	Example
^{ }	Superscript	'text^{superscript}'
_{ }	Subscript	'text_{subscript}'
\bf	Bold font	'\bf text'
\it	Italic font	'\it text'
\sl	Oblique font (rarely available)	'\sl text'
\rm	Normal font	'\rm text'
\fontname{specifier}	Set specifier as the name of a font family to change the font style. You can use this modifier with other modifiers.	'\fontname{Courier} text'
\fontsize{specifier}	Set specifier as a scalar numeric value to change the font size.	'\fontsize{15} text'
\color{specifier}	Set specifier as one of these colors: red, green, yellow, magenta, blue, black, white, gray, darkGreen, orange, or lightBlue.	'\color{magenta} text'
\color[rgb]{specifier}	Set specifier as a three-element RGB triplet to change the font color.	'\color[rgb]{0,0.5,0.5} text'

LaTeX Markup

To use LaTeX markup, set the TickLabelInterpreter property to 'latex'. The displayed text uses the default LaTeX font style. The FontName, FontWeight, and FontAngle properties do not have an effect. To change the font style, use LaTeX markup within the text.

The maximum size of the text that you can use with the LaTeX interpreter is 1200 characters. For multiline text, the maximum size reduces by about 10 characters per line.

Data Types: `char`

Grid Properties

`'GridOverData'` — Draw grid over data plots

`0` (default) | `1`

Draw grid over data plots, specified as a comma-separated pair consisting of `'GridOverData'` and `0` or `1`.

Data Types: `logical`

`'DrawGridToOrigin'` — Draw radial lines within innermost circle

`0` (default) | `1`

Draw radial lines within innermost circle of the polar plot, specified as a comma-separated pair consisting of `'DrawGridToOrigin'` and `0` or `1`.

Data Types: `logical`

`'GridAutoRefinement'` — Increase angle resolution

`0` (default) | `1`

Increase angle resolution in the polar plot, specified as a comma-separated pair consisting of `'GridAutoRefinement'` and `0` or `1`. This property increases angle resolution by doubling the number of radial lines outside each magnitude.

Data Types: `logical`

`'GridWidth'` — Width of grid lines

`0.5000` (default) | positive scalar

Width of grid lines, specified as a comma-separated pair consisting of `'GridWidth'` and a positive scalar.

Data Types: `double`

`'GridVisible'` — Show grid lines

`1` (default) | `0`

Show grid lines, including magnitude circles and angle radii, specified as a comma-separated pair consisting of `'GridVisible'` and `0` or `1`.

Data Types: `logical`

`'GridForegroundColor'` — Color of foreground grid lines

`[0.8000 0.8000 0.8000]` (default) | `'none'` | character vector of color names

Color of foreground grid lines, specified as a comma-separated pair consisting of `'GridForegroundColor'` and an RGB triplet, character vector of color names, or `'none'`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.

- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"red"`	`"r"`	`[1 0 0]`	`"#FF0000"`	
`"green"`	`"g"`	`[0 1 0]`	`"#00FF00"`	
`"blue"`	`"b"`	`[0 0 1]`	`"#0000FF"`	
`"cyan"`	`"c"`	`[0 1 1]`	`"#00FFFF"`	
`"magenta"`	`"m"`	`[1 0 1]`	`"#FF00FF"`	
`"yellow"`	`"y"`	`[1 1 0]`	`"#FFFF00"`	
`"black"`	`"k"`	`[0 0 0]`	`"#000000"`	
`"white"`	`"w"`	`[1 1 1]`	`"#FFFFFF"`	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`"#0072BD"`	
`[0.8500 0.3250 0.0980]`	`"#D95319"`	
`[0.9290 0.6940 0.1250]`	`"#EDB120"`	
`[0.4940 0.1840 0.5560]`	`"#7E2F8E"`	
`[0.4660 0.6740 0.1880]`	`"#77AC30"`	
`[0.3010 0.7450 0.9330]`	`"#4DBEEE"`	
`[0.6350 0.0780 0.1840]`	`"#A2142F"`	

Data Types: `double` | `char`

`'GridBackGroundColor'` — Color of background grid lines
`'w'` (default) | character vector of color names | `'none'`

Color of background grid lines, specified as a comma-separated pair consisting of `'GridBackGroundColor'` and an RGB triplet, character vector of color names, or `'none'`.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (#) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"red"	"r"	[1 0 0]	"#FF0000"	
"green"	"g"	[0 1 0]	"#00FF00"	
"blue"	"b"	[0 0 1]	"#0000FF"	
"cyan"	"c"	[0 1 1]	"#00FFFF"	
"magenta"	"m"	[1 0 1]	"#FF00FF"	
"yellow"	"y"	[1 1 0]	"#FFFF00"	
"black"	"k"	[0 0 0]	"#000000"	
"white"	"w"	[1 1 1]	"#FFFFFF"	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
[0 0.4470 0.7410]	"#0072BD"	
[0.8500 0.3250 0.0980]	"#D95319"	
[0.9290 0.6940 0.1250]	"#EDB120"	
[0.4940 0.1840 0.5560]	"#7E2F8E"	
[0.4660 0.6740 0.1880]	"#77AC30"	
[0.3010 0.7450 0.9330]	"#4DBEEE"	
[0.6350 0.0780 0.1840]	"#A2142F"	

Data Types: `double` | `char`

Marker, Color, Line, and Font Properties

`'Marker'` — Marker symbol
`'none'` (default) | character vector of symbols

Marker symbol, specified as a comma-separated pair consisting of `'Marker'` and either `'none'` or one of the symbols in this table. By default, a line does not have markers. Add markers at selected points along the line by specifying a marker.

Marker	Description	Resulting Marker
"o"	Circle	○
"+"	Plus sign	+
"*"	Asterisk	✳
"."	Point	•

Marker	Description	Resulting Marker
`"x"`	Cross	×
`"_"`	Horizontal line	—
`"|"`	Vertical line	|
`"square"`	Square	□
`"diamond"`	Diamond	◇
`"^"`	Upward-pointing triangle	△
`"v"`	Downward-pointing triangle	▽
`">"`	Right-pointing triangle	▷
`"<"`	Left-pointing triangle	◁
`"pentagram"`	Pentagram	☆
`"hexagram"`	Hexagram	✡
`"none"`	No markers	Not applicable

`'MarkerSize'` — Marker size
`6` (default) | positive value

Marker size, specified as a comma-separated pair consisting of `'MarkerSize'` and a positive value in point units.

Data Types: `double`

`'ColorOrder'` — Colors to use for multiline plots
seven predefined colors (default) | three-column matrix of RGB triplets

Colors to use for multi-line plots, specified as a comma-separated pair consisting of `'ColorOrder'` and a three-column matrix of RGB triplets. Each row of the matrix defines one color in the color order.

Data Types: `double`

`'ColorOrderIndex'` — Next color to use in color order
`1` (default) | positive integer

Next color to use in color order, specified as a comma-separated pair consisting of `'ColorOrderIndex'` and a positive integer. New plots added to the axes use colors based on the current value of the color order index.

Data Types: `double`

`'EdgeColor'` — Color of data lines
`'k'` (default) | RGB triplet vector

Color of data lines, specified as a comma-separated pair consisting of `'EdgeColor'` and a character vector of color names or RGB triplet vector.

RGB triplets and hexadecimal color codes are useful for specifying custom colors.

- An RGB triplet is a three-element row vector whose elements specify the intensities of the red, green, and blue components of the color. The intensities must be in the range `[0,1]`; for example, `[0.4 0.6 0.7]`.
- A hexadecimal color code is a character vector or a string scalar that starts with a hash symbol (`#`) followed by three or six hexadecimal digits, which can range from `0` to `F`. The values are not case sensitive. Thus, the color codes `'#FF8800'`, `'#ff8800'`, `'#F80'`, and `'#f80'` are equivalent.

Alternatively, you can specify some common colors by name. This table lists the named color options, the equivalent RGB triplets, and hexadecimal color codes.

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"red"`	`"r"`	`[1 0 0]`	`"#FF0000"`	
`"green"`	`"g"`	`[0 1 0]`	`"#00FF00"`	
`"blue"`	`"b"`	`[0 0 1]`	`"#0000FF"`	
`"cyan"`	`"c"`	`[0 1 1]`	`"#00FFFF"`	
`"magenta"`	`"m"`	`[1 0 1]`	`"#FF00FF"`	
`"yellow"`	`"y"`	`[1 1 0]`	`"#FFFF00"`	
`"black"`	`"k"`	`[0 0 0]`	`"#000000"`	
`"white"`	`"w"`	`[1 1 1]`	`"#FFFFFF"`	

Here are the RGB triplets and hexadecimal color codes for the default colors MATLAB uses in many types of plots.

RGB Triplet	Hexadecimal Color Code	Appearance
`[0 0.4470 0.7410]`	`"#0072BD"`	
`[0.8500 0.3250 0.0980]`	`"#D95319"`	
`[0.9290 0.6940 0.1250]`	`"#EDB120"`	
`[0.4940 0.1840 0.5560]`	`"#7E2F8E"`	
`[0.4660 0.6740 0.1880]`	`"#77AC30"`	
`[0.3010 0.7450 0.9330]`	`"#4DBEEE"`	
`[0.6350 0.0780 0.1840]`	`"#A2142F"`	

Data Types: `double` | `char`

`'LineStyle'` — Line style of the plot
`'-'` (default) | `'--'` | `':'` | `'-.'` | `'none'`

Line style of the plot, specified as a comma-separated pair consisting of `'LineStyle'` and one of the symbols in the table:

Symbol	Line Style	Resulting Line
'-'	Solid line	———————
'--'	Dashed line	- — — — —
':'	Dotted line	....
'-.'	Dash-dotted line	—·—·—·—·-
'none'	No line	No line

'LineWidth' — Line width of plot
1 (default) | positive scalar | positive vector

Line width of the plot, specified as a comma-separated pair consisting of 'LineWidth' and a positive scalar or vector.

'FontSize' — Font size of text in plot
10 (default) | positive scalar

Font size of text in the plot, specified as a comma-separated pair consisting of 'FontSize' and a positive scalar.

'FontSizeAutoMode' — Set font size
'auto' (default) | 'manual'

Set font size, specified as a comma-separated pair consisting of 'FontSizeAutoMode' and 'auto' or 'manual'.

Data Types: char

showSpan

Show or hide angle span between two markers

Syntax

```
showSpan(p,id1,id2)
showSpan(p,id1,id2,true)
showSpan(p,vis)
showSpan(p)
d = showSpan( ___ )
```

Description

`showSpan(p,id1,id2)` displays the angle span between two angle markers, `id1` and `id2`. The angle span is calculated counterclockwise.

`showSpan(p,id1,id2,true)` automatically reorders the angle markers such that the initial angle span is less than or equal to 180° counterclockwise.

`showSpan(p,vis)` sets angle span visibility by setting `vis` to `true` or `false`.

`showSpan(p)` toggles the angle span display on and off.

`d = showSpan( ___ )` returns angle span details in a structure, `d` using any of the previous syntaxes.

Input Arguments

p — Polar plot
scalar handle

Polar plot, specified as a scalar handle.

`id1,id2` — Cursor or peak marker identifiers
character vector

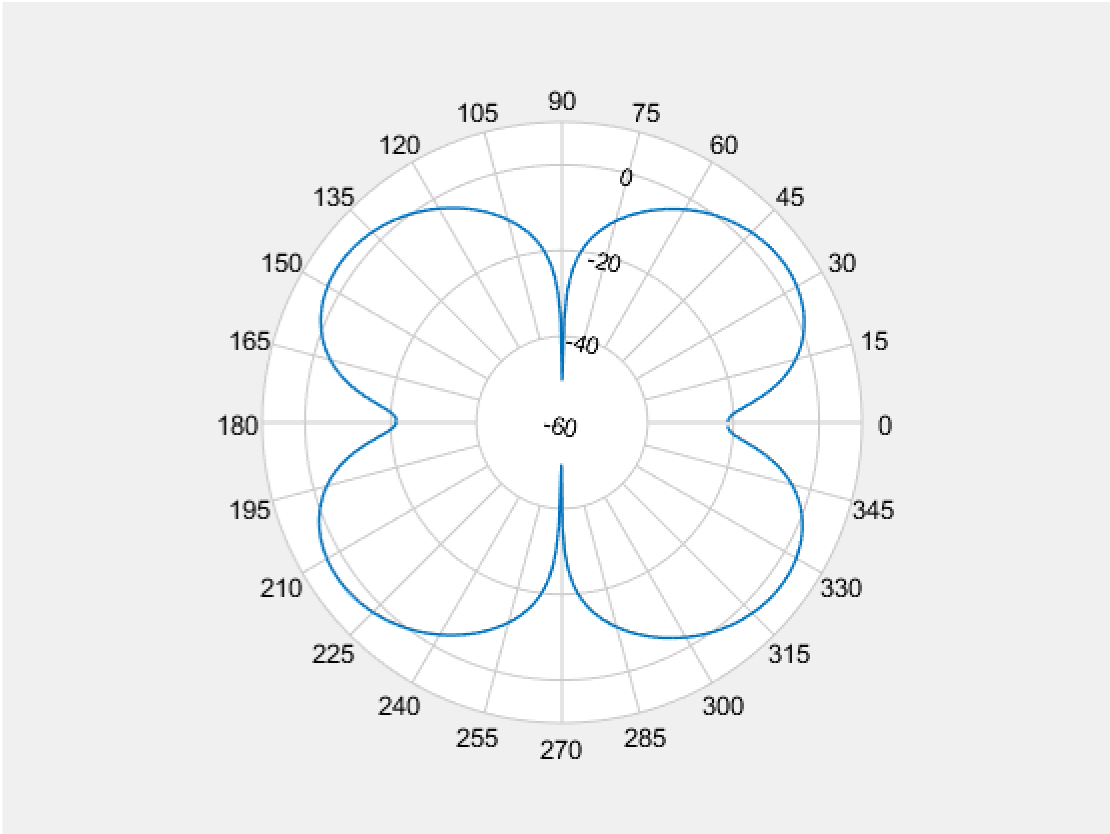
Cursor or peak marker identifiers, specified as character vector. Adding cursors to the polar plot creates cursor marker identifiers. Adding peaks to the polar plot creates peak marker identifiers.

Example: `showspan(p,'C1','C2')`. Displays the angle span between cursors, `C1` and `C2` in polar plot, p.

ExamplesShow Angle Span

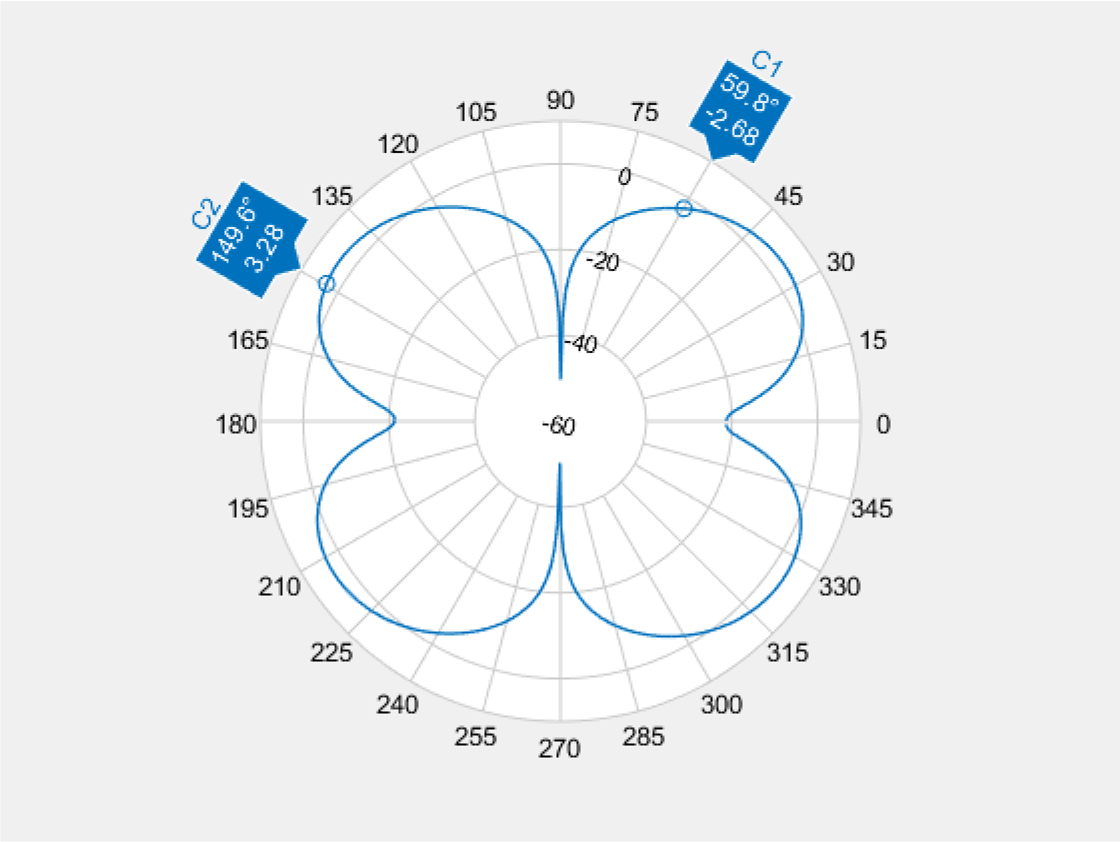
Create a dipole antenna and plot the directivity at 270 MHz.

```
d = dipole;
D = pattern(d,270e6,0,0:1:360);
p = polarpattern(D);
```



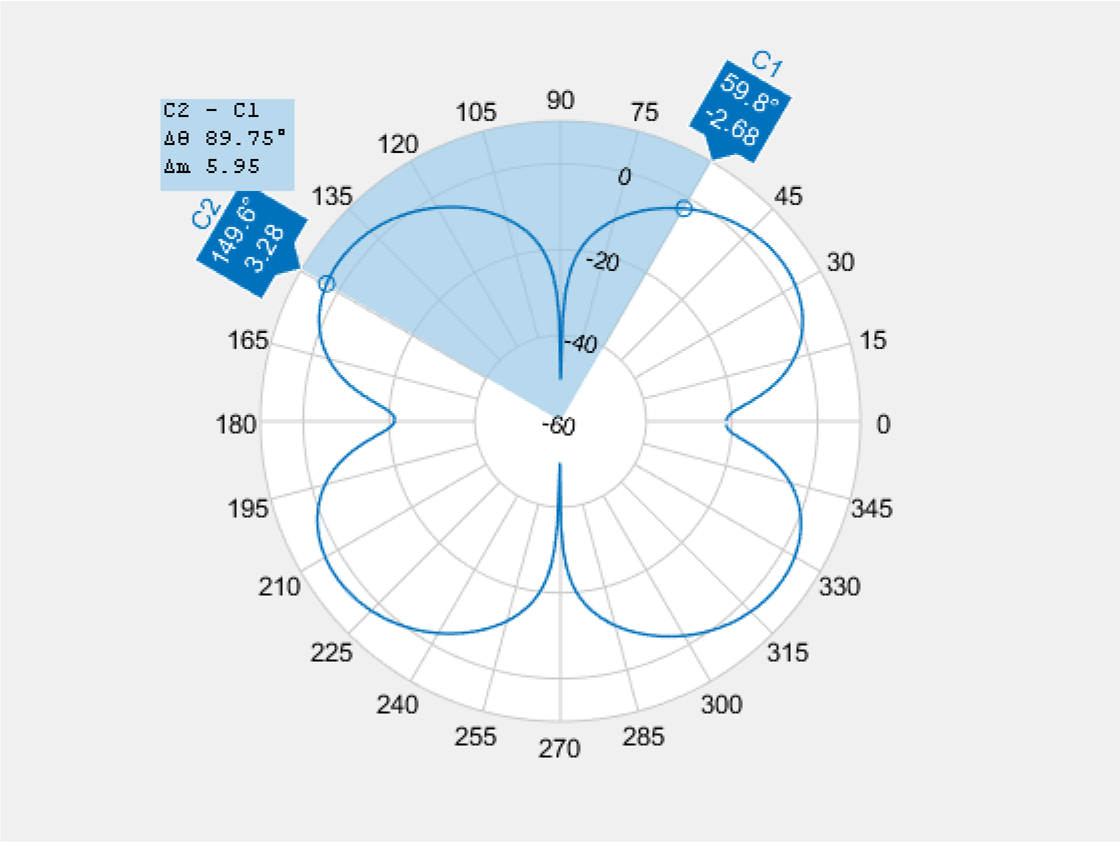
Add cursors to the polar plot at approximately 60 and 150 degrees.

```
addCursor(p,[60 150]);
```



Show the angle span between the two angles.

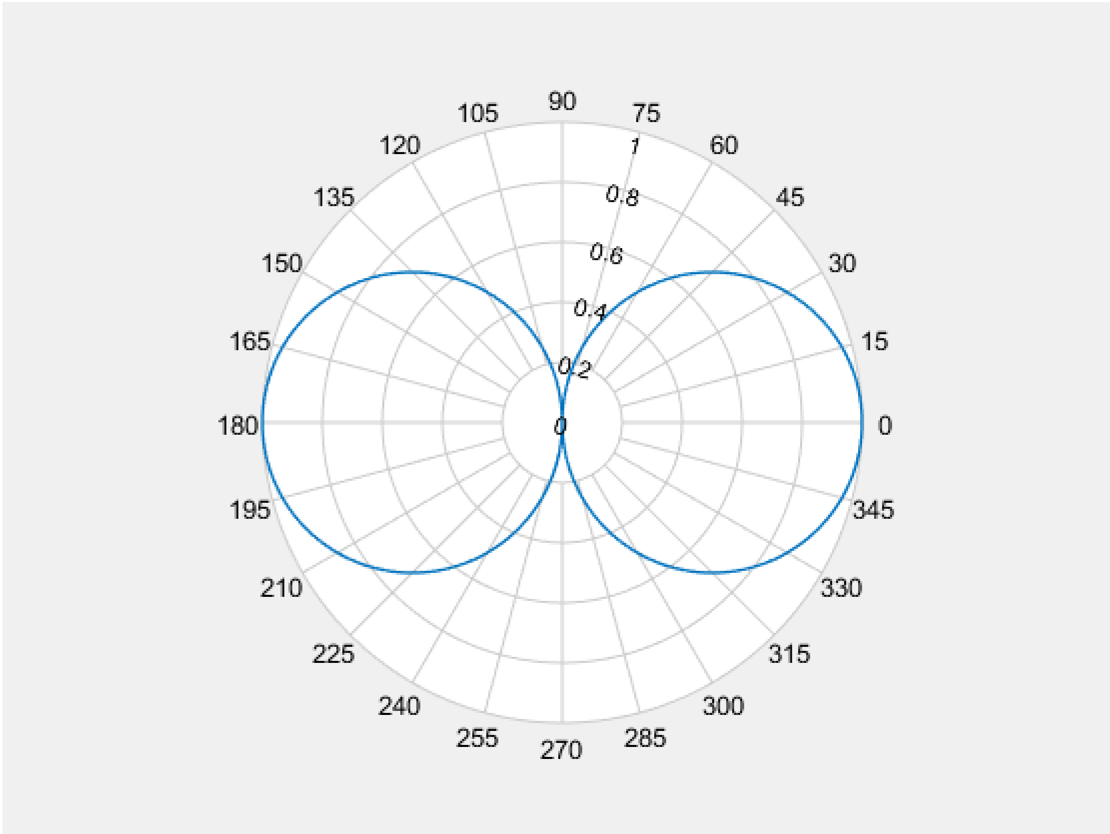
```
showSpan(p,'C1','C2');
```



Show Angle Span for Short-Dipole Antenna

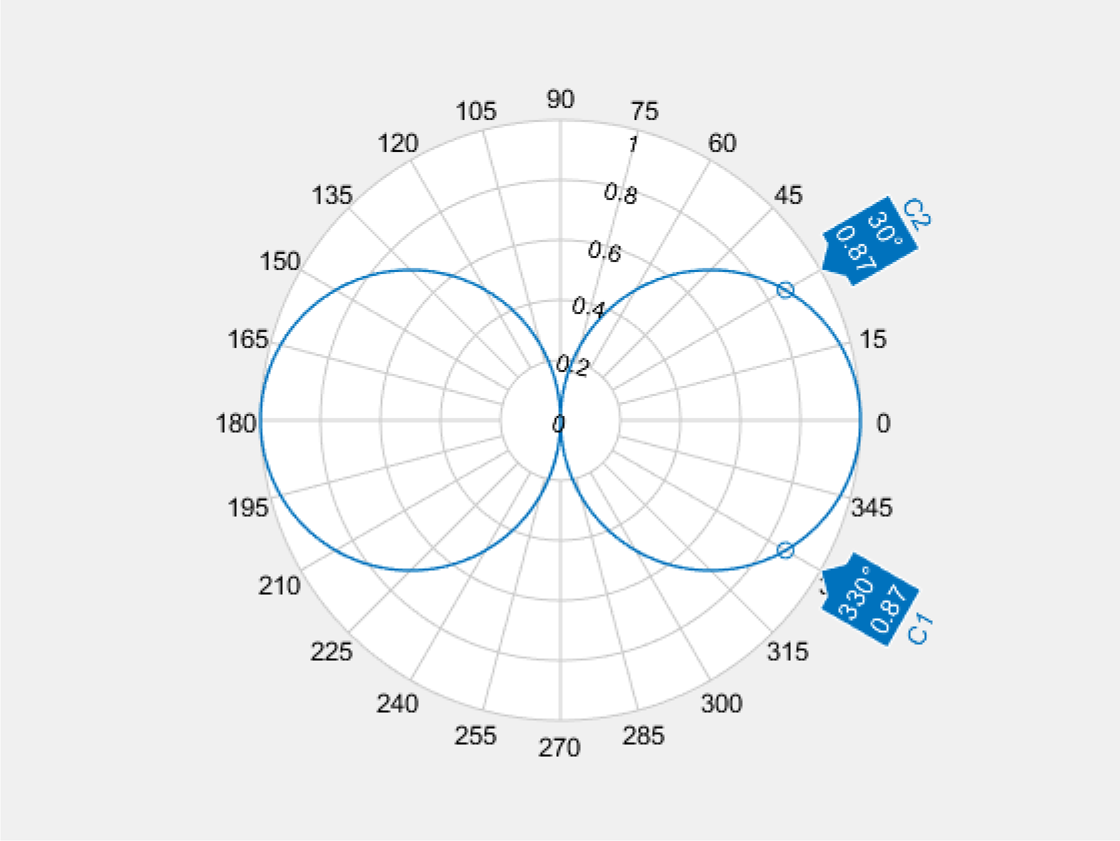
Create a short-dipole antenna element and plot the field values at 250 MHz.

```
antenna = phased.ShortDipoleAntennaElement('FrequencyRange',[100,900]*1e6,...
    'AxisDirection','Y');
angs = [-180:1:180];
fc = 250.0e6;
p = pattern(antenna,250.0e6,angs,0,'CoordinateSystem','polar','Type',...
    'efield','Polarization','H');
P = polarpattern(angs,abs(p));
```



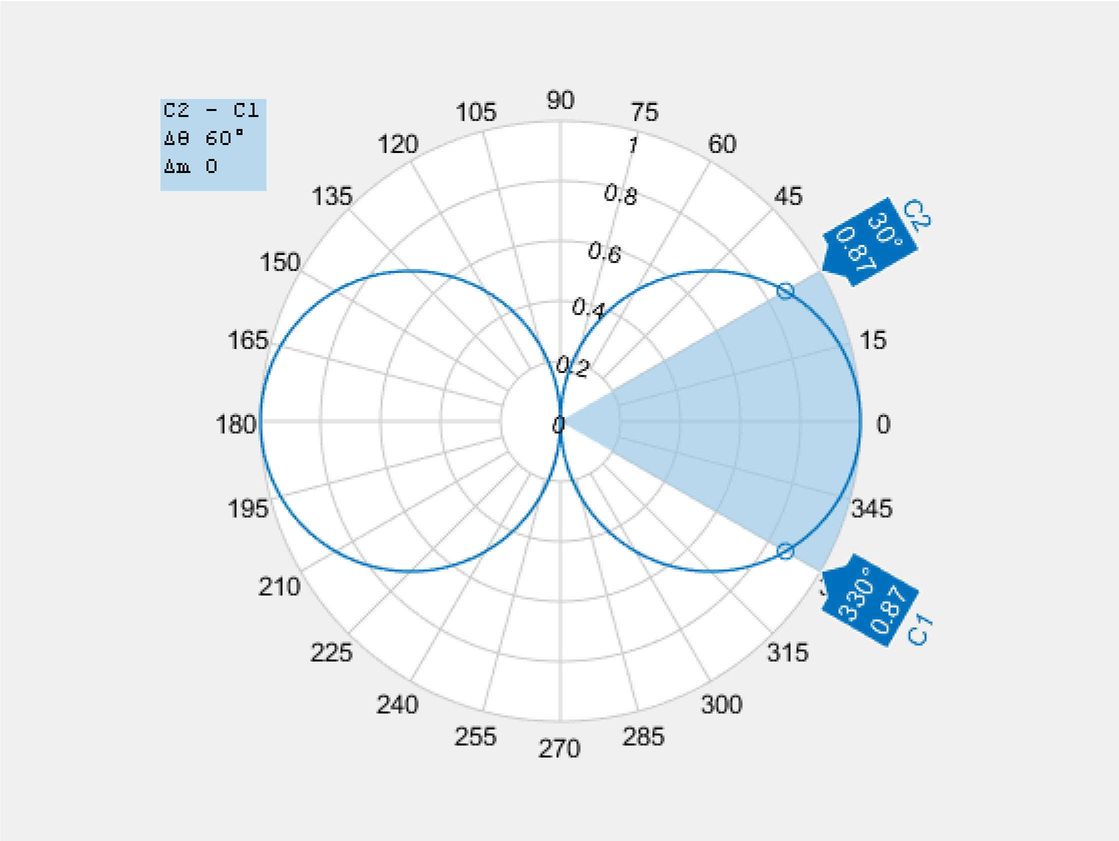
Add cursors to the polar plot at -30° and 30°.

```
addCursor(P,[-30 30]);
```



Show the angle span between the two angles.

```
showSpan(P,'C1','C2');
```



Version History

Introduced in R2016a

See Also

`add` | `addCursor` | `animate` | `createLabels` | `findLobes` | `replace` | `showPeaksTable`

AlphaBetaFilter

Alpha-beta filter for object tracking

Description

The `AlphaBetaFilter` object represents an alpha-beta filter designed for object tracking. Use this tracker for platforms that follow a linear motion model and have a linear measurement model. Linear motion is defined by constant velocity or constant acceleration. Use the filter to predict the future location of an object, to reduce noise for a detected location, or to help associate multiple objects with their tracks.

Creation

Syntax

```
abf = AlphaBetaFilter
abf = AlphaBetaFilter(Name,Value,...)
```

Description

`abf = AlphaBetaFilter` creates an alpha-beta filter for a discrete time, 2-D constant velocity system. The motion model of the filter corresponds to setting the `MotionModel` property to `'2D Constant Velocity'`. In this case, the filter state takes the form `[x; vx; y; vy]`.

`abf = AlphaBetaFilter(Name,Value,...)` specifies the properties of the filter using one or more `Name,Value` pair arguments. Any unspecified properties take default values.

Properties

`MotionModel` — Model of target motion
`'2D Constant Velocity'` (default) | `'1D Constant Velocity'` | `'3D Constant Velocity'` | `'1D Constant Acceleration'` | `'2D Constant Acceleration'` | `'3D Constant Acceleration'`

Model of target motion, specified as a character vector or string. Specifying `1D`, `2D` or `3D` sets the dimensions of the targets motion. Specifying `Constant Velocity` assumes that the target motion has constant velocity at each simulation step. Specifying `Constant Acceleration` assumes that the target motion has constant acceleration at each simulation step.

Data Types: `char` | `string`

`State` — Filter state
scalar | real-valued *M*-element vector

Filter state, specified as a scalar or a real-valued *M*-element vector. A scalar input is extended to an *M*-element vector. The state vector is the concatenated states from each dimension.

The state vectors for each motion model are column vectors:

MotionModel Property	State Vector
'1D Constant Velocity'	[x; vx]
'2D Constant Velocity'	[x; vx; y; vy]
'3D Constant Velocity'	[x; vx; y; vy; z; vz]
'1D Constant Acceleration'	[x; vx; ax]
'2D Constant Acceleration'	[x; vx; ax; y; vy; ay]
'3D Constant Acceleration'	[x; vx; ax; y; vy; ay; z; vz; az]

where, for example, `vx` denotes velocity in the *x*-direction and `ax` denotes acceleration in the *x*-direction.

Example: `[200;0.2;150;0.1;0;0.25]`

Data Types: `double`

`StateCovariance` — State estimation error covariance
M-by-*M* matrix | scalar

State error covariance, specified as an *M*-by-*M* matrix where *M* is the size of the filter state. A scalar input is extended to an *M*-by-*M* matrix. The covariance matrix represents the uncertainty in the filter state.

Example: `eye(6)`

`ProcessNoise` — Process noise covariance
D-by-*D* matrix | scalar

Process noise covariance, specified as a scalar or an *D*-by-*D* matrix where *D* is the dimensionality of motion. For example, if `MotionModel` is `'2D Constant Velocity`, then *D* = 2. A scalar input is extended to an *D*-by-*D* matrix.

Example: `[20 0.1; 0.1 1]`

`MeasurementNoise` — Measurement noise covariance
D-by-*D* matrix | scalar

Measurement noise covariance, specified as a scalar or an *D*-by-*D* matrix where *D* is the dimensionality of motion. For example, if `MotionModel` is `'2D Constant Velocity`, then *D* = 2. A scalar input is extended to an *M*-by-*M* matrix.

Example: `[20 0.1; 0.1 1]`

`Coefficients` — Alpha-beta filter coefficients
scalar | row vector of real values

Alpha-beta filter coefficients, specified as a scalar or row vector of real values. Any scalar input is extended to a row vector. If you specify constant velocity in the `MotionModel` property, the coefficients are `[alpha beta]`. If you specify constant acceleration in the `MotionModel` property, the coefficients are `[alpha beta gamma]`.

Example: `[20 0.1]`

Object Functions

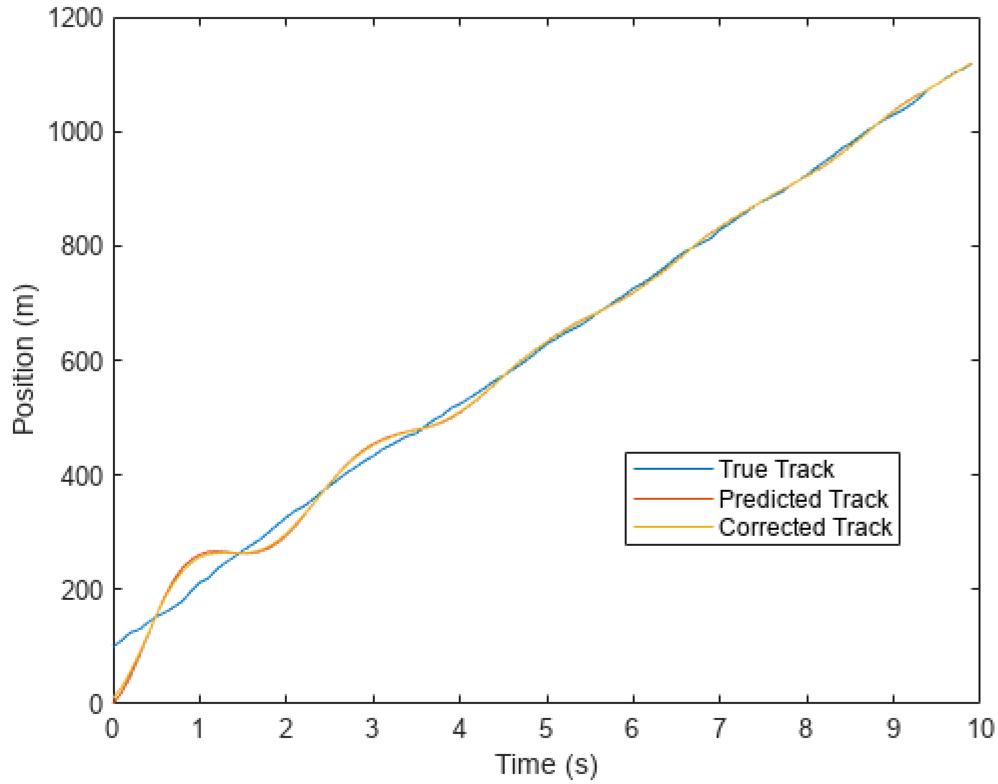
predict	Predict the state and state estimation error covariance
correct	Correct the state and state estimation error covariance
distance	Distances between measurements and predicted measurements
likelihood	Likelihood of measurement
clone	Create identical object

Examples

Track Constant-Velocity Target Using Alpha-Beta Filter

Apply the alpha-beta filter to track a target moving at constant velocity along the x-axis.

```
T = 0.1;
V0 = 100;
N = 100;
plat = phased.Platform('MotionModel','Velocity', ...
    'VelocitySource','Input port','InitialPosition',[100;0;0]);
abfilt = phased.AlphaBetaFilter('MotionModel','1D Constant Velocity');
Z = zeros(1,N);
Zp = zeros(1,N);
Zc = zeros(1,N);
for m = 1:N
    pos = plat(T,[100+20*randn;0;0]);
    Z(m) = pos(1);
    [~,~,Zp(m)] = predict(abfilt,T);
    [~,~,Zc(m)] = correct(abfilt,Z(m));
end
t = (0:N-1)*T;
plot(t,Z,t,Zp,t,Zc)
xlabel('Time (s)')
ylabel('Position (m)')
legend('True Track','Predicted Track','Corrected Track', ...
    'Location','Best')
```

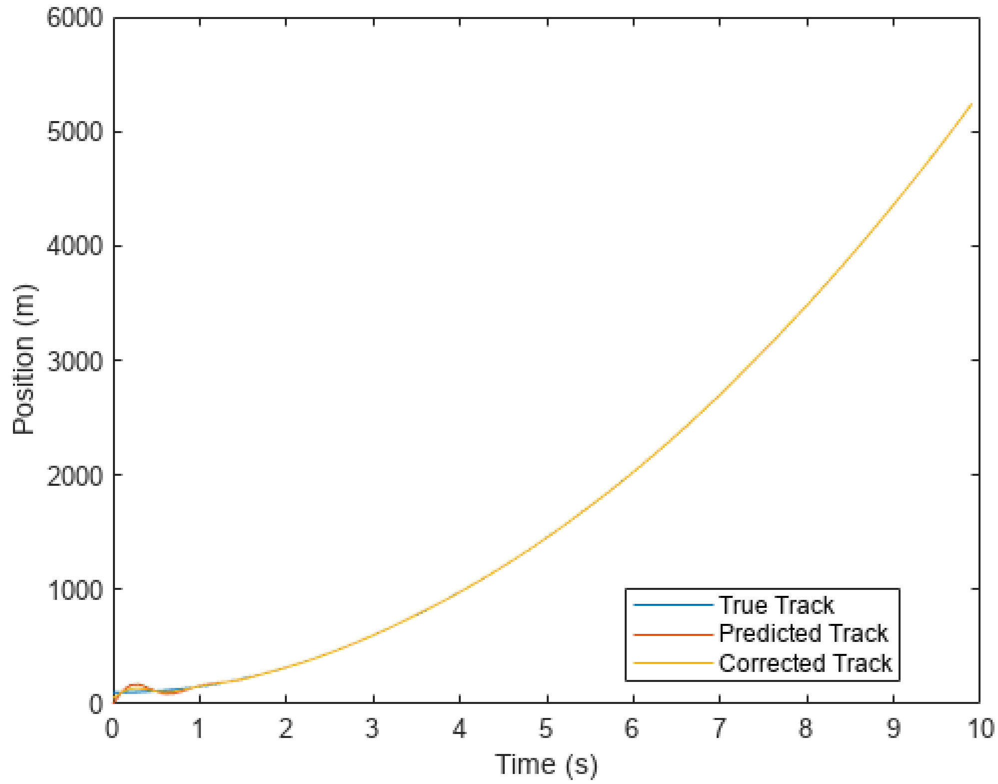


Track Constant-Acceleration Target Using Alpha-Beta Filter

Apply the alpha-beta filter to track a target moving at constant acceleration along the x-axis.

```
T = 0.1;
a0 = 100;
N = 100;
plat = phased.Platform('MotionModel','Acceleration', ...
    'AccelerationSource','Input port','InitialPosition',[100;0;0]);
abfilt = phased.AlphaBetaFilter( ...
    'MotionModel','1D Constant Acceleration', ...
    'Coefficients',[0.5 0.5 0.1]);
Z = zeros(1,N);
Zp = zeros(1,N);
Zc = zeros(1,N);
for m = 1:N
    pos = plat(T,[100+20*randn;0;0]);
    Z(m) = pos(1);
    [~,~,Zp(m)] = predict(abfilt,T);
    [~,~,Zc(m)] = correct(abfilt,Z(m));
end
t = (0:N-1)*T;
plot(t,Z,t,Zp,t,Zc)
xlabel('Time (s)')
ylabel('Position (m)');
```

```
legend('True Track','Predicted Track','Corrected Track', ...
    'Location','Best');
```

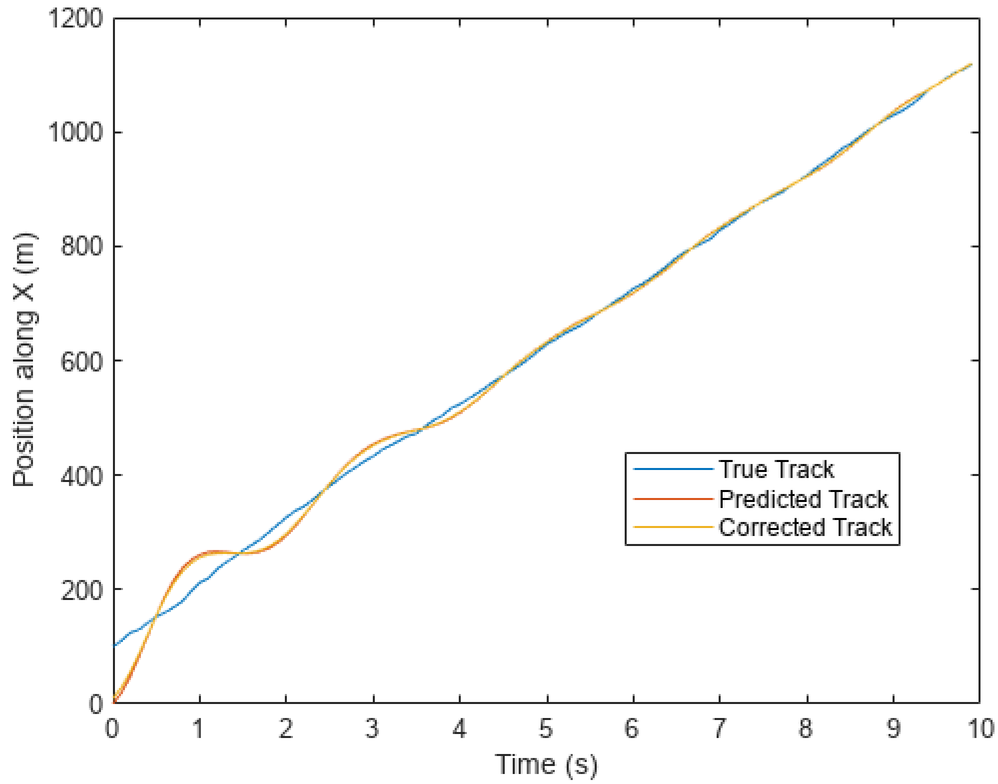


Track Target in 3-D Using Alpha-Beta Filter

Apply the alpha-beta filter to track a target moving at constant velocity in three dimensions.

```
T = 0.1;
V0 = 100;
N = 100;
plat = phased.Platform('MotionModel','Velocity', ...
    'VelocitySource','Input port','InitialPosition',[100;0;0]);
abfilt = phased.AlphaBetaFilter('MotionModel', ...
    '3D Constant Velocity','State',zeros(6,1));
Z = zeros(3,N);
Zp = zeros(3,N);
Zc = zeros(3,N);
for m = 1:N
    Z(:,m) = plat(T,[V0+20*randn;0;0]);
    [~,~,Zp(:,m)] = predict(abfilt,T);
    [~,~,Zc(:,m)] = correct(abfilt,Z(:,m));
end
t = (0:N-1)*T;
plot(t,Z(1,:),t,Zp(1,:),t,Zc(1,:))
xlabel('Time (s)')
```

```
ylabel('Position along X (m)')
legend('True Track','Predicted Track','Corrected Track', ...
    'Location','Best')
```



Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

clone

Copy of alpha-beta tracking filter

Syntax

```
abfilter2 = clone(abfilter)
```

Description

`abfilter2 = clone(abfilter)` creates a copy, `abfilter2`, of the filter object, `abfilter`, with the same property values.

Input Arguments

`abfilter` — Alpha-beta tracking filter
`phased.AlphaBetaFilter` object

Alpha-beta tracking filter, specified as a `phased.AlphaBetaFilter` object.

Output Arguments

`abfilter2` — Copy of alpha-beta tracking filter
`phased.AlphaBetaFilter`

Copy of alpha-beta tracking filter, returned as a `phased.AlphaBetaFilter` object.

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`predict` | `correct` | `distance` | `likelihood`

Objects
`phased.Platform`

correct

Correct the state and state estimation error covariance

Syntax

```
xCorr = correct(abfilter,zMeas)
[xCorr,pCorr] = correct(abfilter,zMeas)
[xCorr,pCorr,zCorr] = correct(abfilter,zMeas)
```

Description

`xCorr = correct(abfilter,zMeas)` returns the corrected the state, `xCorr`, of the tracking filter `abfilter` given the measurement, `zMeas`. Calling `correct` overwrites the internal states of the object.

`[xCorr,pCorr] = correct(abfilter,zMeas)` also returns the corrected state covariance matrix, `pCorr`.

`[xCorr,pCorr,zCorr] = correct(abfilter,zMeas)` also returns the corrected measurement, `zCorr`.

Input Arguments

`abfilter` — Alpha-beta tracking filter
`phased.AlphaBetaFilter` object

Alpha-beta tracking filter, specified as a `phased.AlphaBetaFilter` object. Calling `correct` overwrites the internal states of the object.

`zMeas` — Measurement of tracked object
K-by-1 vector

Measurement of tracked object, specified as a *K*-by-1 vector, where *K* is the size of the measurement.

Output Arguments

`xCorr` — Corrected state of the filter
L-by-1 vector

Corrected state of the filter, returned as an *L*-by-1 vector. The corrected state overrides the value of the `State` property.

`pCorr` — Corrected state covariance of the filter
L-by-*L* matrix

Corrected state covariance of the filter, returned as an *L*-by-*L* matrix. The corrected state covariance overrides the value of the `StateCovariance` property.

`zCorr` — Corrected measurement of the filter
K-by-1 vector

Corrected measurement of the filter, returned as a *K*-by-1 vector.

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`predict` | `distance` | `likelihood` | `clone`

Objects

`phased.Platform`

distance

Distances between measurements and predicted measurements

Syntax

```
dist = distance(abfilter,zMatrix)
```

Description

`dist = distance(abfilter,zMatrix)` computes the distance between one or more predicted measurements given in `zMatrix` and the measurement predicted by the `abfilter` object.

Input Arguments

`abfilter` — Alpha-beta tracking filter
`phased.AlphaBetaFilter` object

Alpha-beta tracking filter, specified as a `phased.AlphaBetaFilter` object.

`zMatrix` — Measurements of tracked objects
N-by-*K* matrix

Measurements of tracked objects, specified as an *N*-by-*K* matrix where *N* is the number of measurements. Each row of the matrix contains a measurement vector. The number of columns, *K*, must match the measurement dimensions of the motion model. This computation takes into account the covariance of the predicted state and the process noise.

Output Arguments

`dist` — Distances between measurements and filter predictions
length-*N* row vector

Distances between measurements and filter predictions, returned as a row vector. Each element corresponds to a distance between the predicted measurement coming from the `abfilter` object and a row of `zMatrix`.

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
predict | correct | likelihood | clone

Objects
phased.Platform

predict

Predict the state and state estimation error covariance

Syntax

```
xPred = predict(abfilter,tstep)
[xPred,pPred] = predict(abfilter,tstep)
[xPred,pPred,zPred] = predict(abfilter,tstep)
```

Description

`xPred = predict(abfilter,tstep)` returns the predicted filter state, `xPred`, of the filter, `abfilter`, after the elapsed time, `tstep`.

`[xPred,pPred] = predict(abfilter,tstep)` also returns the estimated state covariance, `pPred`.

`[xPred,pPred,zPred] = predict(abfilter,tstep)` also returns the predicted measurements, `zPred`.

Input Arguments

`abfilter` — Alpha-beta tracking filter object
`phased.AlphaBetaFilter` object

Alpha-beta tracking filter, specified as a `phased.AlphaBetaFilter` object. Calling `predict` overwrites the internal states of the object.

`tstep` — Time step
positive scalar

Time step for next prediction, specified as a positive scalar. The time step is the interval from the last prediction-correction to the current prediction. Units are in seconds.

Output Arguments

`xPred` — Predicted state of the filter
L-by-1 vector

Predicted state of the filter, returned as an L-by-1 vector where L is the size of the state vector. The predicted state overrides the value of the `State` property.

`pPred` — Predicted state covariance of the filter
L-by-L matrix

Predicted state covariance of the filter, returned as an L-by-L matrix. The predicted state covariance overrides the value of the `StateCovariance` property.

`zPred` — Predicted measurement
K-by-1 vector

Predicted measurement, returned as a *K*-by-1 vector, where *K* is the size of the measurement.

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`correct` | `distance` | `likelihood` | `clone`

Objects

`phased.Platform`

likelihood

Likelihood of measurement

Syntax

```
lk = likelihood(abfilter,zMeas)
```

Description

`lk = likelihood(abfilter,zMeas)` computes the likelihood, `lk`, of the current measurement, `zMeas`, from the filter, `abfilter`.

Input Arguments

`abfilter` — Alpha-beta tracking filter
`phased.AlphaBetaFilter` object

Alpha-beta tracking filter, specified as a `phased.AlphaBetaFilter` object.

`zMeas` — Measurements of tracked object
K-by-1 vector

Measurements of tracked object, specified as a *K*-by-1 vector, where *K* is the size of the measurement.

Output Arguments

`lk` — Likelihood of measurement
scalar

Likelihood of current measurement, returned as a scalar.

Version History

Introduced in R2018b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`predict` | `correct` | `distance` | `clone`

Objects
`phased.Platform`

directivity

Package: `phased`

Directivity of antenna or transducer element

Syntax

```
D = directivity(element,FREQ,ANGLE)
```

Description

`D = directivity(element,FREQ,ANGLE)` returns the "Directivity" on page 1-2132 of the antenna or transducer element, `element`, at frequencies specified by FREQ in direction angles specified by `ANGLE`.

Input Arguments

`element` — Antenna or transducer element

Phased Array System Toolbox System object

Antenna or transducer element, specified as a Phased Array System Toolbox System object.

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

`ANGLE` — Angles for computing directivity

1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Output Arguments

`D` — Directivity

M-by-*L* matrix

Directivity, returned as an *M*-by-*L* matrix. Each row corresponds to one of the *M* angles specified by `ANGLE`. Each column corresponds to one of the *L* frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

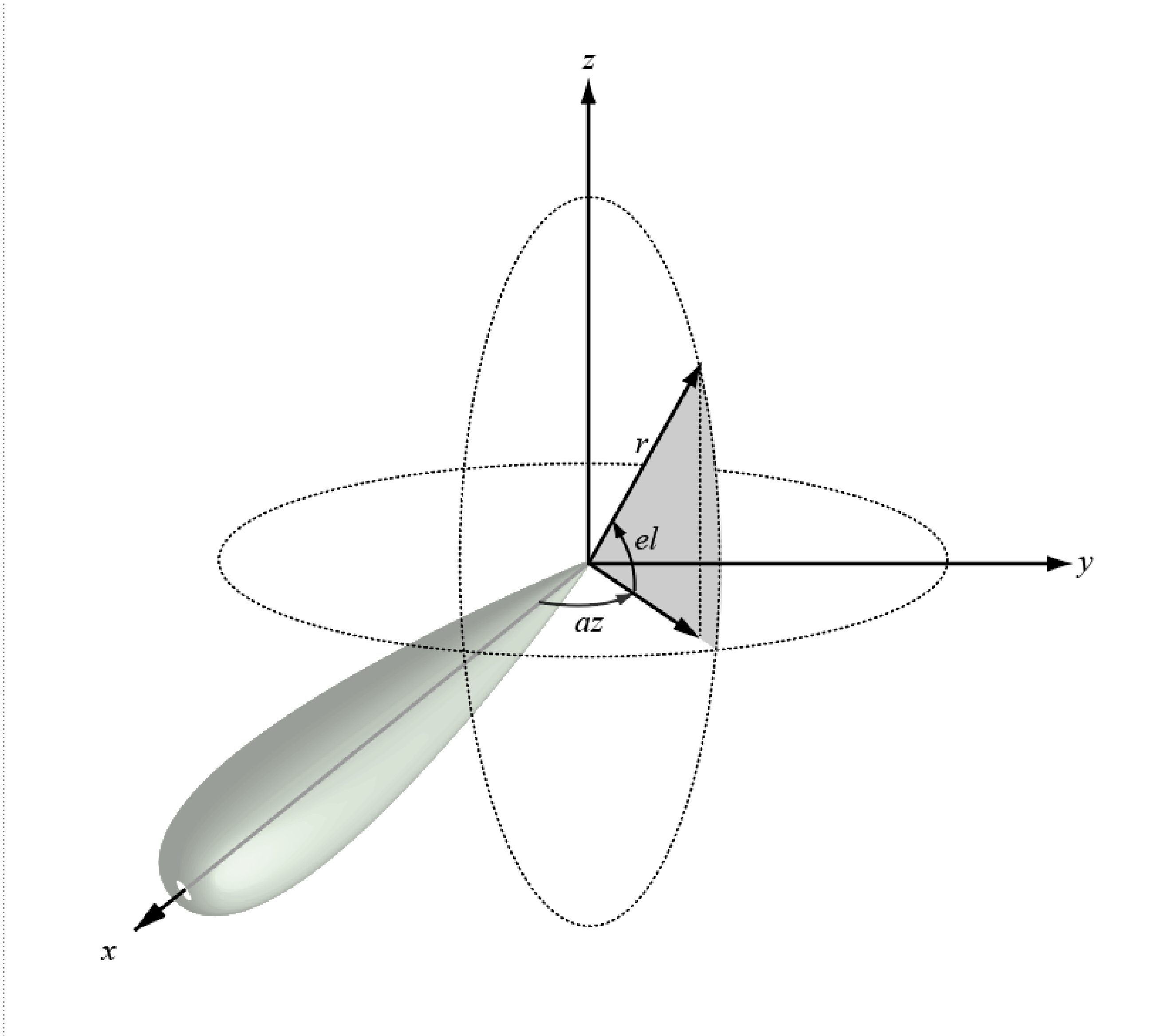
where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on "Element Directivity" and "Array Directivity".

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy*-plane. The angle is positive from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive toward the positive *z*-axis from the *xy*-plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

This figure illustrates the azimuth and elevation angles of a direction vector.



Version History

Introduced in R2019a

See Also

`pattern` | `patternAzimuth` | `patternElevation` | `beamwidth`

beamwidth

Package: phased

Compute and display beamwidth of an array

Syntax

```
beamwidth(array,freq)
beamwidth(array,freq,Name,Value)
[bw,angles] = beamwidth( ___ )
```

Description

`beamwidth(array,freq)` plots the 2-D power pattern (in dB) of the `array` for all azimuth angles at a fixed elevation angle of zero degrees. The plot displays the half-power beamwidth (in degrees) at the frequency specified in `freq` (in Hz) and the angles (in degrees) in azimuth at which the magnitude of the power pattern decreases by 3 dB from the peak of the main beam.

`beamwidth(array,freq,Name,Value)` plots the beamwidth with the specified parameter `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1,Value1,...,NameN,ValueN`).

Example: `beamwidth(array,3e8,'Cut','Elevation')`

`[bw,angles] = beamwidth( ___ )` returns the angular beamwidth `bw` (in degrees). The function also returns the corresponding angle values (in degrees) that mark the beamwidth.

Examples

Plot Beamwidth of Sonar Array

Plot the beamwidth of a sonar array operating at a frequency of 2 kHz when the propagation speed of sound in water is 1500 m/s.

The sonar array consists of a 20-element uniform linear array (ULA). Consider the element of the ULA to be a backbaffled `phased.IsotropicProjector` with a `VoltageResponse` of 100 Volts and with a `FrequencyRange` from 10 Hz to 300 kHz. Create a `phased.ULA` object to model the uniform linear array.

```
projector = phased.IsotropicProjector('BackBaffled',true,...
       'VoltageResponse',100,'FrequencyRange',[10 300000])
```

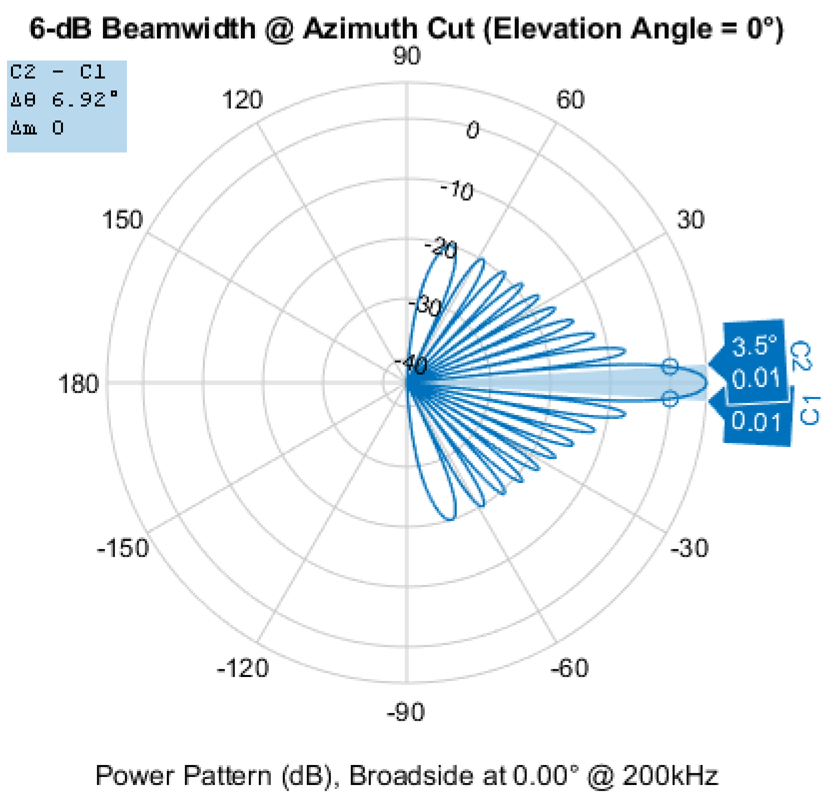
```
projector = 
  phased.IsotropicProjector with properties:

    VoltageResponse: 100
     FrequencyRange: [10 300000]
        BackBaffled: true
```

```
myArray = phased.ULA('Element',projector,'NumElements',20,...
    'ElementSpacing',1500/200e3/2)
```

```
myArray = 
  phased.ULA with properties:

           Element: [1x1 phased.IsotropicProjector]
       NumElements: 20
    ElementSpacing: 0.0037
         ArrayAxis: 'y'
             Taper: 1
```

Using the `beamwidth` function, calculate and plot the 6 dB beamwidth of the sonar array.

```
beamwidth(myArray,200e3,'dBDown',6,'PropagationSpeed',1500)
```




```
ans = 6.9200
```

Calculate Beamwidth and Angles of Uniform Linear Array (ULA)

Calculate the half-power beamwidth and angles of a 20-element uniform linear array (ULA) of cosine antenna elements.

Create a `phased.CosineAntennaElement` object with the `'CosinePower'` exponents set to 1.5.

```
myAnt = phased.CosineAntennaElement
```

```
myAnt = 
  phased.CosineAntennaElement with properties:

    FrequencyRange: [0 1.0000e+20]
       CosinePower: [1.5000 1.5000]
```

Create a `phased.ULA` object to model a 20-element ULA of cosine antenna elements. These elements are spaced at 0.5 meters on the azimuth plane.

```
array = phased.ULA('Element',myAnt,'NumElements',20)
```

```
array = 
  phased.ULA with properties:

           Element: [1x1 phased.CosineAntennaElement]
       NumElements: 20
    ElementSpacing: 0.5000
         ArrayAxis: 'y'
             Taper: 1
```

Compute the beamwidth and angles of the array when it is operating at 3e8 Hz. Specify the beamwidth to be computed along the elevation plane.

```
[BW,Ang] = beamwidth(array,3e8,'Cut','Elevation')
```

```
BW = 74.8200

Ang = 1×2

  -37.4100   37.4100
```

Input Arguments

`array` — Array of sensor elements
Phased Array System Toolbox System object

Array of sensor elements, specified as one of the following System objects:

- `phased.ULA`
- `phased.URA`
- `phased.UCA`
- `phased.ConformalArray`
- `phased.HeterogeneousULA`
- `phased.HeterogeneousURA`
- `phased.HeterogeneousConformalArray`

`freq` — Frequency used to calculate beamwidth
scalar in Hz

Frequency used to calculate the beamwidth, specified as a scalar in Hz.

Example: 3e8

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose Name *in quotes.*

Example: beamwidth(array,3e8,'Cut','Azimuth','CutAngle',45) plots the beamwidth of the array that is operating at a frequency of 0.3 GHz, with the slice direction set to 'Azimuth', and the cut angle set to 45 degrees.

Cut — Slice direction in azimuth-elevation space
'Azimuth' (default) | 'Elevation'

The slice direction in azimuth-elevation space along which the beamwidth is computed, specified as the comma-separated pair consisting of 'Cut' and 'Azimuth' for the azimuth plane, and 'Cut' and 'Elevation' for the elevation plane.

CutAngle — Angle for plane to get required 2-D cut
0 (default) | scalar

Corresponding angle (in degrees) for the plane to get the required 2-D cut, specified as the comma-separated pair consisting of 'CutAngle' and a scalar. If 'Cut' is specified as 'Azimuth', then 'CutAngle' (Elevation) should lie between [−90, 90] degrees. If 'Cut' is specified as 'Elevation', then 'CutAngle' (Azimuth) should lie between [−180, 180] degrees.

Data Types: double

dBDown — Power value from peak of main lobe
3 (default) | positive scalar

Power value (in dB) from the peak of the main lobe, specified as the comma-separated pair consisting of 'dBDown' and a positive scalar. The default value is 3 dB, which translates to half-power beamwidth. To calculate first-null beamwidth, specify the 'dBDown' value as Inf.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

PropagationSpeed — Propagation speed
3×10^8 m/s (speed of light) (default) | positive scalar

Propagation speed, specified as the comma-separated pair consisting of 'PropagationSpeed' and a positive scalar (in m/s).

Data Types: double

Weights — Weights applied to array
length-*N* column vector

Weights applied to the array of sensor elements, specified as the comma-separated pair consisting of 'Weights' and a length-*N* column vector, where *N* is the number of elements in the array.

Data Types: `double`

Output Arguments

`bw` — Angular beamwidth
scalar in degrees

Angular beamwidth of the array of sensor elements, returned as a scalar in degrees.

Data Types: `double`

`angles` — Angle values of beamwidth
1-by-2 vector in degrees

Angle values of the beamwidth, returned as a 1-by-2 vector. The two elements in the vector [a_{min}, a_{max}] define the beamwidth `bw` as $a_{max}-a_{min}$.

Version History

Introduced in R2020b

See Also

Objects
`phased.ULA` | `phased.URA` | `phased.UCA` | `phased.ConformalArray` | `phased.HeterogeneousULA` | `phased.HeterogeneousURA` | `phased.HeterogeneousConformalArray`

beamwidth

Package: phased

Compute and display beamwidth of a subarray

Syntax

```
beamwidth(subarray,freq)
beamwidth(subarray,freq,Name,Value)
[bw,angles] = beamwidth( ___ )
```

Description

`beamwidth(subarray,freq)` plots the 2-D power pattern (in dB) of the `subarray` for all azimuth angles at a fixed elevation angle of zero degrees. The plot displays the half-power beamwidth (in degrees) at the frequency specified in `freq` (in Hz) and the angles (in degrees) in azimuth at which the magnitude of the power pattern decreases by 3 dB from the peak of the main beam.

`beamwidth(subarray,freq,Name,Value)` computes and plots the beamwidth with the specified parameter `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1,Value1,...,NameN,ValueN`).

Example: `beamwidth(subarray,5e8,'Cut','Elevation')`

`[bw,angles] = beamwidth( ___ )` returns the angular beamwidth `bw` (in degrees). The function also returns the corresponding angle values (in degrees) of the beamwidth.

Examples

Plot Beamwidth of Rectangular Lattice Array

Plot the beamwidth of a rectangular lattice array composed of two uniform rectangular arrays. Consider the antenna elements of the array to be cosine antenna elements.

First, construct a `phased.CosineAntennaElement` object.

```
myAnt = phased.CosineAntennaElement
```

```
myAnt = 
  phased.CosineAntennaElement with properties:

    FrequencyRange: [0 1.0000e+20]
       CosinePower: [1.5000 1.5000]
```

Next, construct a 5-by-5 uniform rectangular array by creating a `phased.URA` object.

```
myArray = phased.URA([5 5],[0.5 0.5],'Element',myAnt,...
    'ElementSpacing',[0.15 0.15])
```

```
myArray = 
  phased.URA with properties:
```

```
          Element: [1x1 phased.CosineAntennaElement]
             Size: [5 5]
   ElementSpacing: [0.1500 0.1500]
          Lattice: 'Rectangular'
      ArrayNormal: 'x'
            Taper: 1
```

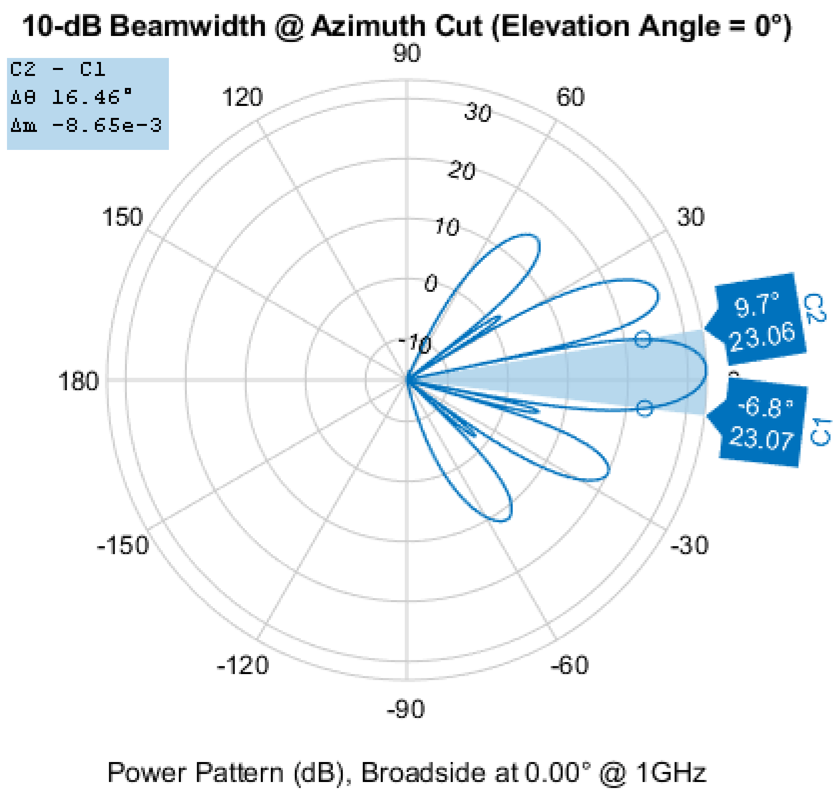
Use two of these 5-by-5 uniform rectangular arrays to construct a 5-by-10 rectangular lattice. Construct the lattice using the `phased.ReplicatedSubarray` object.

```
myRSA = phased.ReplicatedSubarray('Subarray',myArray,...
'Layout','Rectangular','GridSize',[1 2],...
'GridSpacing','Auto','SubarraySteering','Phase')
```

```
myRSA =
  phased.ReplicatedSubarray with properties:

                 Subarray: [1x1 phased.URA]
                   Layout: 'Rectangular'
                 GridSize: [1 2]
              GridSpacing: 'Auto'
         SubarraySteering: 'Phase'
    PhaseShifterFrequency: 300000000
      NumPhaseShifterBits: 0
```

Now visualize the 10dB beamwidth of the obtained lattice across the azimuth plane (0 degrees elevation). The subarray is phase steered toward 24 degrees azimuth. Assume the operating frequency of the array to be 1 GHz.

```
stv = phased.SteeringVector('SensorArray',myRSA);
beamwidth(myRSA,1e9,'dBDown',10,'SteerAngle',24,'Weights',stv(1e9,24))
```




```
ans = 16.4600
```

Calculate Beamwidth and Angles of Two ULAs

Calculate the 3 dB beamwidth of a 10-element uniform linear array (ULA) composed of two 5-element ULAs across the azimuth plane and at 0 degrees elevation. By default, the antenna elements are isotropic. Assume the operating frequency of the array to be 500MHz.

```
myArray = phased.ULA('NumElements',5)
```

```
myArray = 
  phased.ULA with properties:

           Element: [1x1 phased.IsotropicAntennaElement]
       NumElements: 5
    ElementSpacing: 0.5000
         ArrayAxis: 'y'
             Taper: 1
```

```
myRSA = phased.ReplicatedSubarray('Subarray',myArray,...
'GridSize',[1 2])
```

```
myRSA = 
  phased.ReplicatedSubarray with properties:
```

```
            Subarray: [1x1 phased.ULA]
              Layout: 'Rectangular'
            GridSize: [1 2]
         GridSpacing: 'Auto'
    SubarraySteering: 'None'
```

```
[BW,Ang] = beamwidth(myRSA,5e8)
```

```
BW = 6.1200

Ang = 1×2

   -3.0600    3.0600
```

Input Arguments

subarray — Subarray of sensor elements
Phased Array System Toolbox System object

Subarray of sensor elements, specified as one of the following System objects:

- `phased.PartitionedArray`
- `phased.ReplicatedSubarray`

freq — Frequency used to calculate beamwidth
scalar in Hz

Frequency used to calculate the beamwidth, specified as a scalar in Hz.

Example: `5e8`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `beamwidth(subarray,5e8,'Cut','Azimuth','CutAngle',45)` plots the beamwidth of the subarray that is operating at a frequency of 0.5 GHz, with the slice direction set to `'Azimuth'`, and the cut angle set to 45 degrees.

Cut — Slice direction in azimuth-elevation space
`'Azimuth'` (default) | `'Elevation'`

The slice direction in azimuth-elevation space along which the beamwidth is computed, specified as the comma-separated pair consisting of `'Cut'` and `'Azimuth'` for the azimuth plane, and `'Cut'` and `'Elevation'` for the elevation plane.

CutAngle — Angle for plane to get required 2-D cut
0 (default) | scalar

Corresponding angle (in degrees) for the plane to get the required 2-D cut, specified as the comma-separated pair consisting of 'CutAngle' and a scalar. If 'Cut' is specified as 'Azimuth', then 'CutAngle' (Elevation) should lie between [−90, 90] degrees. If 'Cut' is specified as 'Elevation', then 'CutAngle' (Azimuth) should lie between [−180, 180] degrees.

Data Types: double

dBDown — Power value from peak of main lobe
3 (default) | Inf | positive scalar

Power value (in dB) from the peak of the main lobe, specified as the comma-separated pair consisting of 'dBDown' and a positive scalar. The default value is 3 dB, which translates to half-power beamwidth. To calculate the first-null beamwidth, specify the 'dBDown' value as Inf.

Data Types: single | double | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64

PropagationSpeed — Propagation speed
3×10^8 m/s (speed of light) (default) | positive scalar

Propagation speed, specified as the comma-separated pair consisting of 'PropagationSpeed' and a positive scalar (in m/s).

Data Types: double

Weights — Weights applied to array
length-*N* column vector

Weights applied to the array of sensor elements, specified as the comma-separated pair consisting of 'Weights' and a length-*N* column vector, where *N* is the number of elements in the array.

Data Types: double

SteerAngle — Subarray steering angle
[0; 0] (default) | scalar | length-2 column vector

Subarray steering angle (in degrees), specified as the comma-separated pair consisting of 'SteerAngle' and a scalar or a length-2 column vector. If the steering angle is a scalar, the value represents the azimuth angle and the elevation angle is assumed to be 0. If the steering angle is a vector, the angle is specified in the form of [AzimuthAngle; ElevationAngle].

Dependencies

This parameter is applicable when you set the SubarraySteering property of subarray object to either 'Phase' or 'Time'.

Data Types: double

ElementWeights — Weights applied to each element in subarray
matrix of all ones (default) | matrix | cell array

Weights applied to each element in the subarray, specified as the comma-separated pair consisting of 'ElementWeights' and a matrix or a cell array.

For a ReplicatedSubarray object, ElementWeights must be a *NSE*-by-*N* matrix, where *NSE* is the number of elements in each individual subarray and *N* is the number of subarrays. Each column in ElementWeights specifies the weights for the elements in the corresponding subarray.

For a PartitionedArray object, if the individual subarrays have the same number of elements, ElementWeights must be an *NSE*-by-*N* matrix, where *NSE* is the number of elements in each individual subarray and *N* is the number of subarrays.

Each column in the WS property of the subarray object specifies the weights for the elements in the corresponding subarray. If subarrays in the PartitionedArray object have different number of elements, ElementWeights can be one of the following:

- *NSE*-by-*N* matrix -- *NSE* indicates the number of elements in the largest subarray and *N* is the number of subarrays.
- 1-by-*N* cell array -- *N* is the number of subarrays and each cell contains a column vector whose length is the same as the number of elements of the corresponding subarray.

If WS is a matrix, the first *K* entries in each column specify the weights for the elements in the corresponding subarray. *K* is the number of elements in the corresponding subarray. If WS is a cell array, each cell in the array is a column vector specifying the weights for the elements in the corresponding subarray.

Dependencies

This parameter is applicable when you set the SubarraySteering property of subarray object to 'Custom'.

Data Types: double

Output Arguments

bw — Angular beamwidth
scalar in degrees

Angular beamwidth of the subarray, returned as a scalar in degrees.

Data Types: double

angles — Angle values of beamwidth
1-by-2 vector in degrees

Angle values of the beamwidth, returned as a 1-by-2 vector. The two elements in the vector [a_{min}, a_{max}] define the beamwidth bw as $a_{max}-a_{min}$.

Version History

Introduced in R2020b

See Also

Objects
phased.PartitionedArray | phased.ReplicatedSubarray

beamwidth

Package: phased

Compute and display beamwidth of sensor element pattern

Syntax

```
beamwidth(element,freq)
beamwidth(element,freq,Name,Value)
[bw,angles] = beamwidth( ___ )
```

Description

`beamwidth(element,freq)` plots the 2-D power pattern (in dB) of the `element` for all azimuth angles at an elevation angle of zero degrees. The plot displays the half-power beamwidth (in degrees) at the frequency specified in `freq` (in Hz) and the angles (in degrees) in azimuth at which the magnitude of the power pattern decreases by 3 dB from the peak of the main beam.

`beamwidth(element,freq,Name,Value)` plots the beamwidth with the specified parameter `Name` set to the specified `Value`. You can specify additional name-value pair arguments in any order as (`Name1,Value1,...,NameN,ValueN`).

Example: `beamwidth(element,1e9,'Cut','Elevation)`

`[bw,angles] = beamwidth( ___ )` returns the angular beamwidth `bw` (in degrees). The function also returns the corresponding angle values (in degrees) that mark the beamwidth.

Examples

Plot Beamwidth of Isotropic Antenna

Plot the beamwidth for an isotropic antenna at 1 GHz.

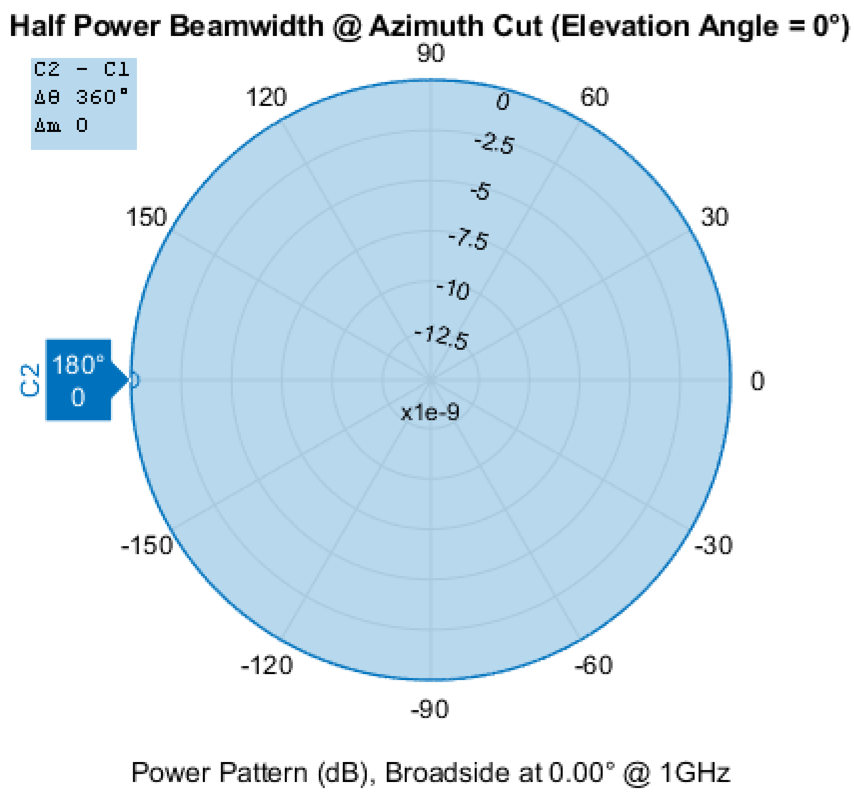
Create an isotropic antenna using a `phased.IsotropicAntennaElement` object.

```
antenna = phased.IsotropicAntennaElement('FrequencyRange',[800e6 1.2e9])
```

```
antenna = 
  phased.IsotropicAntennaElement with properties:

    FrequencyRange: [800000000 1.2000e+09]
       BackBaffled: false
```

Using the `beamwidth` function, plot the half-power (3 dB) beamwidth for the antenna. Use a frequency value of 1 GHz.

```
beamwidth(antenna, 1e9)
```




```
ans = 360
```

Calculate Beamwidth and Angles of Cosine Antenna Element

Calculate the half-power beamwidth and angles of a cosine antenna element.

Create a `phased.CosineAntennaElement` object with the `'CosinePower'` exponents set to 10.

```
myAnt = phased.CosineAntennaElement('CosinePower',[10 10])
```

```
myAnt = 
  phased.CosineAntennaElement with properties:

    FrequencyRange: [0 1.0000e+20]
       CosinePower: [10 10]
```

Compute the beamwidth and angles of the antenna element when it is operating at 1 GHz. Set `'dbDown'` to 3 dB.

```
[BW,Ang] = beamwidth(myAnt,1e9,'dbDown',3)
```

```
BW = 29.9600
```

```
Ang = 1×2
```

```
  -14.9800   14.9800
```

Input Arguments

`element` — Antenna or microphone element
Phased Array System Toolbox System object

Antenna or microphone element, specified as a Phased Array System Toolbox System object.

`freq` — Frequency used to calculate beamwidth
scalar in Hz

Frequency used to calculate the beamwidth, specified as a scalar in Hz.

Example: `1e9`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `beamwidth(element,1e9,'Cut','Azimuth','CutAngle',45)` plots the beamwidth of the antenna element that is operating at a frequency of 1 GHz, with the slice direction set to `'Azimuth'`, and the cut angle set to 45 degrees.

`Cut` — Slice direction in azimuth-elevation space
`'Azimuth'` (default) | `'Elevation'`

The slice direction in azimuth-elevation space along which the beamwidth is computed, specified as the comma-separated pair consisting of `'Cut'` and `'Azimuth'` for the azimuth plane, and `'Cut'` and `'Elevation'` for the elevation plane.

`CutAngle` — Angle for plane to get required 2-D cut
`0` (default) | scalar

Corresponding angle (in degrees) for the plane to get the required 2-D cut, specified as the comma-separated pair consisting of `'CutAngle'` and a scalar. If `'Cut'` is specified as `'Azimuth'`, then `'CutAngle'` (Elevation) should lie between [−90, 90] degrees. If `'Cut'` is specified as `'Elevation'`, then `'CutAngle'` (Azimuth) should lie between [−180, 180] degrees.

Data Types: `double`

`dBDown` — Power value from peak of main lobe
`3` (default) | `Inf` | positive scalar

Power value from the peak of the main lobe, specified as the comma-separated pair consisting of `'dBDown'` and a positive scalar. The default value of 3 dB is equivalent to half-power beamwidth. To calculate the first-null beamwidth, specify the `'dBDown'` value as `Inf`. Units are in dB.

Data Types: `double`

Output Arguments

bw — Angular beamwidth
scalar in degrees

Angular beamwidth of the sensor element, returned as a scalar in degrees.

Data Types: `double`

angles — Angle values of beamwidth
1-by-2 vector in degrees

Angle values of the beamwidth, returned as a 1-by-2 vector. The two elements in the vector [a_{min}, a_{max}] define the beamwidth `bw` as $a_{max}-a_{min}$.

Version History
Introduced in R2020b

See Also

Functions
`directivity` | `isPolarizationCapable` | `pattern` | `patternAzimuth` | `patternElevation`

isPolarizationCapable

Package: `phased`

Antenna element polarization capability

Syntax

`capflag = isPolarizationCapable(element)`

Description

`capflag = isPolarizationCapable(element)` returns a Boolean value, `capflag`, indicating whether the antenna `element` supports polarization. An antenna element supports polarization if it can create or respond to polarized fields.

Input Arguments

`element` — Antenna or transducer element
Phased Array System Toolbox System object

Antenna or transducer element, specified as a Phased Array System Toolbox System object.

Output Arguments

`capflag` — Polarization-capability flag
`true` | `false`

Polarization-capability flag returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not.

Version History

Introduced in R2019a

See Also

Functions
`directivity` | `beamwidth` | `pattern` | `patternAzimuth` | `patternElevation`

pattern

Package: phased

Plot antenna or transducer element directivity and patterns

Syntax

```
pattern(element,FREQ)
pattern(element,FREQ,AZ)
pattern(element,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

pattern(element,FREQ) plots the 3-D array directivity pattern (in dBi) for the element specified in element. The operating frequency is specified in FREQ. You can use this function to display the patterns for antennas that support polarization.

pattern(element,FREQ,AZ) plots the element directivity pattern at the specified azimuth angle.

pattern(element,FREQ,AZ,EL) plots the element directivity pattern at specified azimuth and elevation angles.

pattern(___ ,Name,Value) plots the element pattern with additional options specified by one or more Name,Value pair arguments.

[PAT,AZ_ANG,EL_ANG] = pattern(___) returns the element pattern in PAT. The AZ_ANG output contains the coordinate values corresponding to the rows of PAT. The EL_ANG output contains the coordinate values corresponding to the columns of PAT. If the 'CoordinateSystem' parameter is set to 'uv', then AZ_ANG contains the *U* coordinates of the pattern and EL_ANG contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

Input Arguments

element — Antenna or transducer element
Phased Array System Toolbox System object

Antenna or transducer element, specified as a Phased Array System Toolbox System object.

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `CoordinateSystem,'polar',Type,'directivity'`

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: char

Type — Displayed pattern type
'directivity' (default) | 'efield' | 'power' | 'powerdb'

Displayed pattern type, specified as the comma-separated pair consisting of 'Type' and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

PlotStyle — Plotting style
'overlay' (default) | 'waterfall'

Plotting style, specified as the comma-separated pair consisting of 'Plotstyle' and either 'overlay' or 'waterfall'. This parameter applies when you specify multiple frequencies in FREQ in 2-D plots. You can draw 2-D plots by setting one of the arguments AZ or EL to a scalar.

Data Types: char

Polarization — Polarization type
'combined' (default) | 'H' | 'V'

Polarization type, specified as the comma-separated pair consisting of 'Polarization' and either 'combined', 'H', or 'V'. If Polarization is 'combined', the horizontal and vertical polarization patterns are combined. If Polarization is 'H', only the horizontal polarization is displayed. If Polarization is 'V', only the vertical polarization is displayed.

Dependencies

To enable this property, set the element argument to an antenna that supports polarization: phased.CrossedDipoleAntennaElement, phased.ShortDipoleAntennaElement, or phased.CustomAntennaElement, and then set the 'Type' name-value pair to 'efield', 'power', or 'powerdb'.

Data Types: char | string

Output Arguments

PAT — Element pattern
N-by-*M* real-valued matrix

Element pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of PAT correspond to the azimuth angles in the vector specified by EL_ANG. The columns correspond to the elevation angles in the vector specified by AZ_ANG.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in AZ. The columns of PAT correspond to the values in AZ_ANG. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in EL. The rows of PAT correspond to the values in EL_ANG. Units are in degrees.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

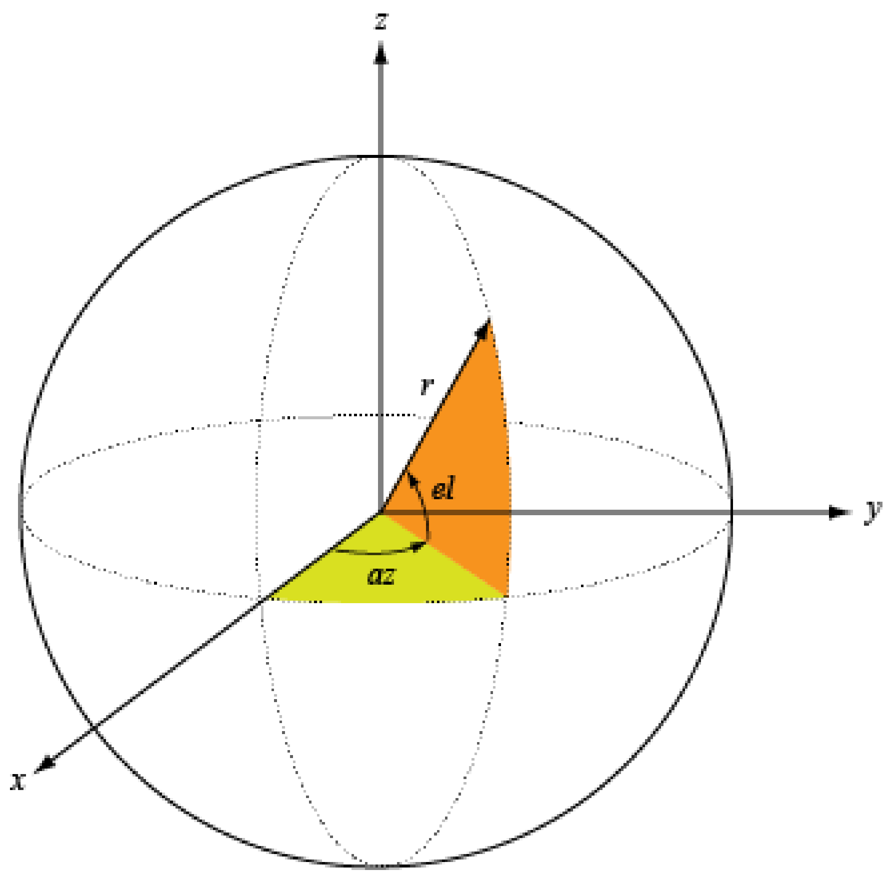
Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and its orthogonal projection onto the *xy*-plane. The angle is positive when going from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180° and 180° degrees, inclusive. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy*-plane. Elevation angles lie between –90° and 90° degrees, inclusive.



Algorithms

Convert plotResponse to Pattern

For antenna, transducer, and array System objects, the `pattern` function replaces the `plotResponse` function. In addition, two new simplified functions exist just to draw 2-D azimuth and elevation pattern plots. These functions are `azimuthPattern` and `elevationPattern`.

The following table is a guide for converting your code from using `plotResponse` to `pattern`. Notice that some of the inputs have changed from *input arguments* to *Name-Value* pairs and conversely. The general `pattern` method syntax is

```
pattern(H,FREQ,AZ,EL,'Name1','Value1',...,'NameN','ValueN')
```

plotResponse Inputs	plotResponse Description	pattern Inputs
`H` argument	Antenna, microphone, or array System object.	`H` argument (no change)
`FREQ` argument	Operating frequency.	`FREQ` argument (no change)
`V` argument	Propagation speed. This argument is used only for arrays.	`'PropagationSpeed'` name-value pair. This parameter is only used for arrays.

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr>
<td>'Format' and 'RespCut' name-value pairs</td>
<td>These options work together to let you create a plot in angle space (line or polar style) or UV space. They also determine whether the plot is 2-D or 3-D. This table shows you how to create different types of plots using plotResponse.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'RespCut' to 'Az' or 'El'. Set 'Format' to 'line' or 'polar'.

Set the display axis using either the 'AzimuthAngles' or 'ElevationAngles' name-value pairs.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'RespCut' to '3D'. Set 'Format' to 'line' or 'polar'.

Set the display axis using both the 'AzimuthAngles' and'ElevationAngles' name-value pairs.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'RespCut' to'U'. Set 'Format' to 'UV'. Set the display range using the 'UGrid' name-value pair.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'RespCut' to'3D'. Set 'Format' to</td></tr>
</table>
</td>
<td>'CoordinateSystem' name-value pair used together with the AZ and EL input arguments.

'CoordinateSystem' has the same options as the plotResponse method 'Format'name-value pair, except that 'line' is now named 'rectangular'. The table shows how to create different types of plots using pattern.
<table>
<tr><th>Display space</th><th></th></tr>
<tr><td>Angle space (2D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify either AZ or EL as a scalar.</td></tr>
<tr><td>Angle space (3D)</td><td>Set 'CoordinateSystem' to 'rectangular' or 'polar'. Specify both AZ and EL as vectors.</td></tr>
<tr><td>UV space (2D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space scalar.</td></tr>
<tr><td>UV space (3D)</td><td>Set 'CoordinateSystem' to 'uv'. Use AZ to specify a U-space vector. Use EL to specify a V-space vector.</td></tr>
</table>
If you set CoordinateSystem to 'uv', enter the UV grid values using AZ and EL.</td>
</tr>
</table>

<table>
<tr><th>plotResponse Inputs</th><th>plotResponse Description</th><th>pattern Inputs</th></tr>
<tr><td></td><td><table><tr><th>Display space</th><th></th></tr><tr><td></td><td>'UV'. Set the display range using both the 'UGrid' and 'VGrid' name-value pairs.</td></tr></table></td><td></td></tr>
<tr><td>'CutAngle' name-value pair</td><td>Constant angle at to take an azimuth or elevation cut. When producing a 2-D plot and when 'RespCut' is set to 'Az' or 'El', use 'CutAngle' to set the slice across which to view the plot.</td><td>No equivalent name-value pair. To create a cut, specify either AZ or EL as a scalar, not a vector.</td></tr>
<tr><td>'NormalizeResponse' name-value pair</td><td>Normalizes the plot. When 'Unit' is set to 'dbi', you cannot specify 'NormalizeResponse'.</td><td>Use the 'Normalize' name-value pair. When 'Type' is set to 'directivity' you cannot specify 'Normalize'.</td></tr>
<tr><td>'OverlayFreq' name-value pair</td><td>Plot multiple frequencies on the same 2-D plot. Available only when 'Format' is set to 'line' or 'uv' and 'RespCut' is not set to '3D'. The value true produces an overlay plot and the value false produces a waterfall plot.</td><td>'PlotStyle' name-value pair plots multiple frequencies on the same 2-D plot.

The values 'overlay' and 'waterfall' correspond to 'OverlayFreq' values of true and false. The option 'waterfall' is allowed only when 'CoordinateSystem' is set to 'rectangular' or 'uv'.</td></tr>
<tr><td>'Polarization' name-value pair</td><td>Determines how to plot polarized fields. Options are 'None', 'Combined', 'H', or 'V'.</td><td>'Polarization' name-value pair determines how to plot polarized fields. The 'None' option is removed. The options 'Combined', 'H', or 'V' are unchanged.</td></tr>
<tr><td>'Unit' name-value pair</td><td>Determines the plot units. Choose 'db', 'mag', 'pow', or 'dbi', where the default is 'db'.</td><td>'Type' name-value pair, uses equivalent options with different names

<table><tr><th>plotResponse</th><th>pattern</th></tr><tr><td>'db'</td><td>'powerdb'</td></tr><tr><td>'mag'</td><td>'efield'</td></tr><tr><td>'pow'</td><td>'power'</td></tr><tr><td>'dbi'</td><td>'directivity'</td></tr></table></td></tr>
<tr><td>'Weights' name-value pair</td><td>Array element tapers (or weights).</td><td>'Weights' name-value pair (no change).</td></tr>
<tr><td>'AzimuthAngles' name-value pair</td><td>Azimuth angles used to display the antenna or array response.</td><td>AZ argument</td></tr>
</table>

plotResponse Inputs	plotResponse Description	pattern Inputs
'ElevationAngles' name-value pair	Elevation angles used to display the antenna or array response.	EL argument
'UGrid' name-value pair	Contains *U* coordinates in *UV*-space.	AZ argument when 'CoordinateSystem' name-value pair is set to 'uv'
'VGrid' name-value pair	Contains *V*-coordinates in *UV*-space.	EL argument when 'CoordinateSystem' name-value pair is set to 'uv'

Version History

Introduced in R2019a

See Also

directivity | patternAzimuth | patternElevation | beamwidth

patternAzimuth

Package: `phased`

Plot antenna or transducer element directivity and pattern versus azimuth

Syntax

```
patternAzimuth(element,FREQ)
patternAzimuth(element,FREQ,EL)
patternAzimuth(element,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(element,FREQ)` plots the 2-D element directivity pattern versus azimuth (in dBi) for the element `element` at zero-degrees elevation angle. The argument `FREQ` specifies the operating frequency.

`patternAzimuth(element,FREQ,EL)`, in addition, plots the 2-D element directivity pattern versus azimuth (in dBi) at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(element,FREQ,EL,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input argument.

Input Arguments

`element` — Antenna or transducer element
Phased Array System Toolbox System object

Antenna or transducer element, specified as a Phased Array System Toolbox System object.

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Azimuth,[-90:90],Type,'directivity'`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

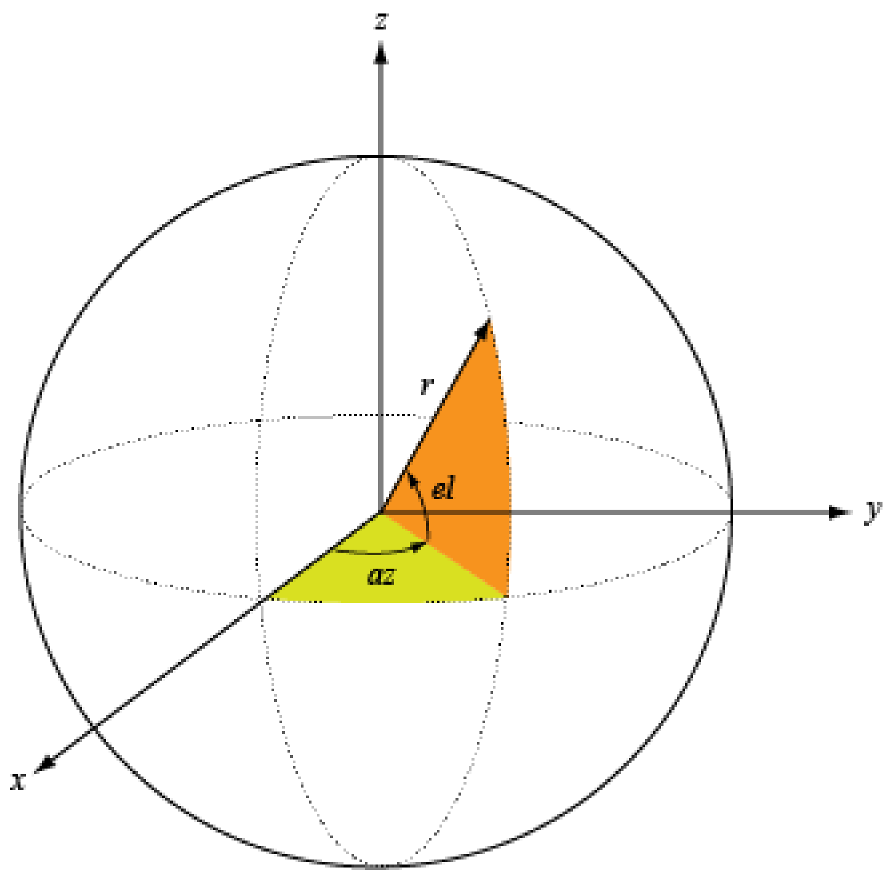
Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and its orthogonal projection onto the *xy*-plane. The angle is positive when going from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180° and 180° degrees, inclusive. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy*-plane. Elevation angles lie between –90° and 90° degrees, inclusive.



Version History

Introduced in R2019a

See Also

`directivity` | `pattern` | `patternElevation` | `beamwidth`

patternElevation

Package: `phased`

Plot antenna or transducer element directivity and pattern versus elevation

Syntax

```
patternElevation(element,FREQ)
patternElevation(element,FREQ,AZ)
patternElevation(element,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

`patternElevation(element,FREQ)` plots the 2-D element directivity pattern versus elevation (in dBi) for the element `element` at zero-degrees azimuth angle. The argument `FREQ` specifies the operating frequency.

`patternElevation(element,FREQ,AZ)`, in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by `AZ`. When `AZ` is a vector, multiple overlaid plots are created.

`patternElevation(element,FREQ,AZ,Name,Value)` plots the element pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternElevation( ___ )` returns the element pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Elevation'` parameter and the `AZ` input argument.

Input Arguments

`element` — Antenna or transducer element
Phased Array System Toolbox System object

Antenna or transducer element, specified as a Phased Array System Toolbox System object.

`FREQ` — Frequency for computing directivity and pattern
positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or the `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

AZ — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `Azimuth,[-90:90],Type,'directivity'`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-*P* real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-*P* real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

Output Arguments

PAT — Element directivity or pattern
P-by-*N* real-valued matrix

Element directivity or pattern, returned as an *P*-by-*N* real-valued matrix. The dimension *P* is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension *N* is the number of azimuth angles determined by the `AZ` argument.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

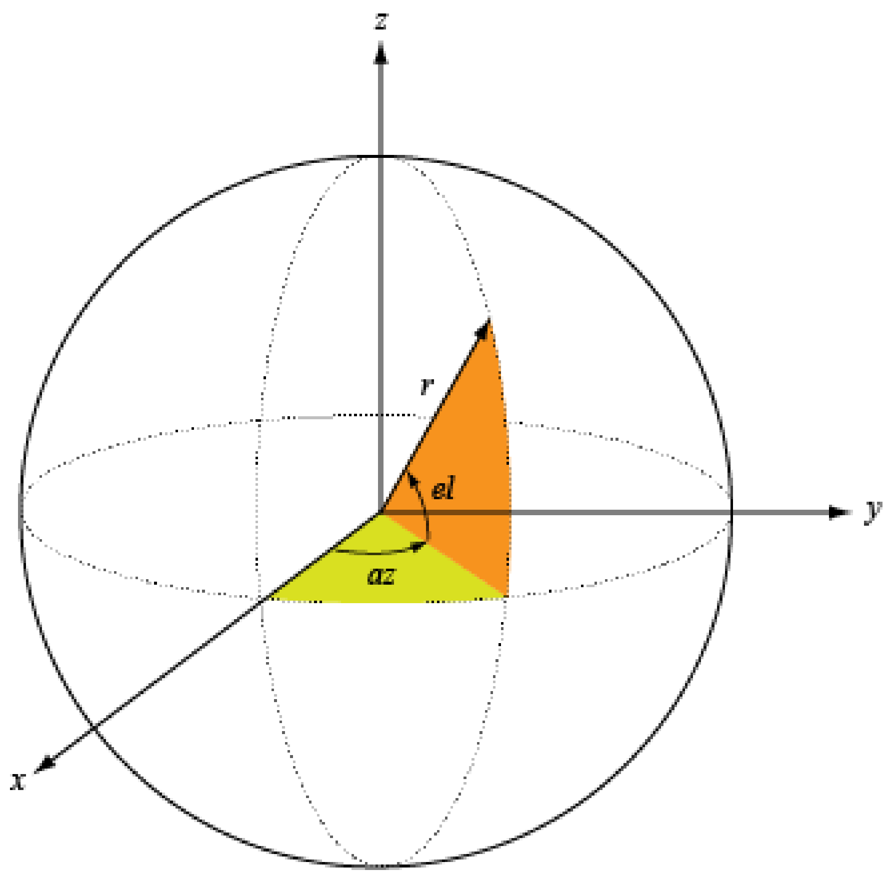
Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and its orthogonal projection onto the *xy*-plane. The angle is positive when going from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180° and 180° degrees, inclusive. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy*-plane. Elevation angles lie between –90° and 90° degrees, inclusive.



Version History

Introduced in R2019a

See Also

directivity | pattern | patternAzimuth | beamwidth

plotSpectrum

Package: `phased`

Plot spatial spectrum

Syntax

```
plotSpectrum(estimator)
plotSpectrum(estimator,Name,Value)
hl = plotSpectrum( ___ )
```

Description

`plotSpectrum(estimator)` plots the spatial spectrum resulting from the most recent execution of the `estimator` object.

`plotSpectrum(estimator,Name,Value)` plots the spatial spectrum with additional options specified by one or more `Name,Value` pair arguments.

`hl = plotSpectrum( ___ )` returns the line handle of the spectrum plot in the figure.

Input Arguments

`estimator` — Spectrum estimator
System object

Spectrum estimator, specified as a Phased Array System Toolbox System object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'NormalizedResponse',true,'Unit','pow'`

`NormalizeResponse` — Plot normalized response
`false` (default) | `true`

Option to enable plotting of normalized response, specified as `false` or `true`. Set this value to `true` to plot the normalized spectrum. Set this value to `false` to plot the spectrum without normalization.

Data Types: `logical`

`Title` — Title of plot figure
`''` (default) | character vector

Title of plot figure, specified as a character vector.

Example: `'Beamscan Spectrum'`

Data Types: `char`

`Unit` — Plot units
`'db'` (default) | `'mag'` | `'pow'`

Plot units, specified as

- `'db'` – decibels
- `'mag'` – magnitude
- `'pow'` – power

.

Output Arguments

`hl` — Line handle
line handle

Plot line, returned as a handle.

Version History
Introduced in R2012a

collectPlaneWave

Simulate received plane waves at array

Syntax

```
Y = collectPlaneWave(array,X,ang)
Y = collectPlaneWave(array,X,ang,freq)
Y = collectPlaneWave(array,X,ang,freq,c)
```

Description

`Y = collectPlaneWave(array,X,ang)` returns the sum `Y` of received signals `X` arriving at the sensor `array` from the directions specified in `ang`.

`Y = collectPlaneWave(array,X,ang,freq)` also specifies the incoming signal carrier frequency `freq`.

`Y = collectPlaneWave(array,X,ang,freq,c)` also specifies the signal propagation speed `c`.

Examples

Simulate Received Signals at ULA

Simulate two received plane-wave random signals at a 4-element ULA. The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
array = phased.ULA(4);
y = collectPlaneWave(array,randn(4,2),[10 30],100e6,physconst('LightSpeed'))
```

```
y = 4×4 complex

   0.7430 - 0.3705i   0.8433 - 0.1314i   0.8433 + 0.1314i   0.7430 + 0.3705i
   0.8418 + 0.4308i   0.5632 + 0.1721i   0.5632 - 0.1721i   0.8418 - 0.4308i
  -2.4817 + 0.9157i  -2.6683 + 0.3175i  -2.6683 - 0.3175i  -2.4817 - 0.9157i
   1.0724 - 0.4748i   1.1895 - 0.1671i   1.1895 + 0.1671i   1.0724 + 0.4748i
```

Input Arguments

`array` — Phased array System object
System object

Phased array, specified as a Phased Array System Toolbox System object.

`X` — Incoming signals
P-by-*M* complex-valued matrix

Incoming signals in each direction, specified as a *P*-by-*M* complex-valued matrix. *M* is the number of signals impinging on the array. Each column of X represents an individual incoming signal. *P* is the number of signal samples. Each signal is assumed to arrive from the direction specified by the `ang` argument.

Example: `[1,5;2,10;3,10]`

Data Types: `double`
Complex Number Support: Yes

ang — Arrival directions of incoming signals
1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Arrival directions of incoming signals, specified as a 1-by-*M* vector or a 2-by-*M* matrix, where *M* is the number of incoming signals. Each column specifies the direction of arrival of the corresponding signal in X. If ANG is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the incoming signal `[az;el]`. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°.

If ANG is a 1-by-*M* vector, then each entry represents a set of azimuth angles, with the elevation angles assumed to be zero. Angle units are in degrees.

The azimuth angle is the angle between the *x*-axis and the projection of the arrival direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, the azimuth angle is positive.

The elevation angle is the angle between the arrival direction vector and the *xy*-plane. When measured toward the *z* axis, the elevation angle is positive.

Example: `[20,30;15,25]`

Data Types: `double`

`freq` — Signal carrier frequency
3e8 (default) | positive scalar

Signal carrier frequency, specified as a positive scalar. Units are in Hz.

Data Types: `double`

c — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second.

Example: `physconst('LightSpeed')`

Data Types: `double`

Output Arguments

Y — Combined received signals
P-by-*N* complex-valued matrix

Combined received signals at each array element, returned as a *P*-by-*N* complex-valued matrix where *N* is the number of array elements. Each column of Y contains the combined received signals at the corresponding array element. *P* is the number of signal samples.

Data Types: `double`

Complex Number Support: Yes

Algorithms

The `collectPlaneWave` modulates a signal by a phase delay caused by the direction of arrival. This is called the *phase-shift approximation*. The method does not account for the response of individual elements in the array.

For further details, see Van Trees [1] .

Version History

Introduced in R2021a

References

[1] Van Trees, H. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`uv2azel` | `phitheta2azel`

directivity

Compute array directivity

Syntax

```
D = directivity(array,FREQ,ANGLE)
directivity( ___ ,Name,Value)
```

Description

`D = directivity(array,FREQ,ANGLE)` returns the "Directivity (dBi)" on page 1-1743 of an array of antenna or microphone elements, `array`, at frequencies specified by `FREQ` and in angles of direction specified by `ANGLE`.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`directivity( ___ ,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

Examples

Directivity of Uniform Linear Array

Compute the directivities of two different uniform linear arrays (ULA). One array consists of isotropic antenna elements and the second array consists of cosine antenna elements. In addition, compute the directivity when the first array is steered in a specified direction. For each case, calculate the directivities for a set of seven different azimuth directions all at zero degrees elevation. Set the frequency to 800 MHz.

Array of isotropic antenna elements

First, create a 10-element ULA of isotropic antenna elements spaced 1/2-wavelength apart.

```
c = physconst('LightSpeed');
fc = 3e8;
lambda = c/fc;
ang = [-30,-20,-10,0,10,20,30; 0,0,0,0,0,0,0];
myAnt1 = phased.IsotropicAntennaElement;
myArray1 = phased.ULA(10,lambda/2,'Element',myAnt1);
```

Compute the directivity.

```
d = directivity(myArray1,fc,ang,'PropagationSpeed',c)
```

```
d = 7×1

   -6.9886
   -6.2283
   -6.5176
```

```
   10.0011
   -6.5176
   -6.2283
   -6.9886
```

Array of cosine antenna elements

Next, create a 10-element ULA of cosine antenna elements spaced 1/2-wavelength apart.

```
myAnt2 = phased.CosineAntennaElement('CosinePower',[1.8,1.8]);
myArray2 = phased.ULA(10,lambda/2,'Element',myAnt2);
```

Compute the directivity.

```
d = directivity(myArray2,fc,ang,'PropagationSpeed',c)
```

```
d = 7×1

   -1.9838
    0.0529
    0.4968
   17.2548
    0.4968
    0.0529
   -1.9838
```

The directivity of the cosine ULA is greater than the directivity of the isotropic ULA because of the larger directivity of the cosine antenna element.

Steered array of isotropic antenna elements

Finally, steer the isotropic antenna array to 30 degrees in azimuth and compute the directivity.

```
w = steervec(getElementPosition(myArray1)/lambda,[30;0]);
d = directivity(myArray1,fc,ang,'PropagationSpeed',c,...
    'Weights',w)
```

```
d = 7×1

 -297.2705
  -13.9783
   -9.5713
   -6.9897
   -4.5787
   -2.0536
   10.0000
```

The directivity is greatest in the steered direction.

Input Arguments

array — Phased array
Phased Array System Toolbox System object

Phased array, specified as a Phased Array System Toolbox System object.

FREQ — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

ANGLE — Angles for computing directivity
1-by-*M* real-valued row vector | 2-by-*M* real-valued matrix

Angles for computing directivity, specified as a 1-by-*M* real-valued row vector or a 2-by-*M* real-valued matrix, where *M* is the number of angular directions. Angle units are in degrees. If `ANGLE` is a 2-by-*M* matrix, then each column specifies a direction in azimuth and elevation, `[az;el]`. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°.

If `ANGLE` is a 1-by-*M* vector, then each entry represents an azimuth angle, with the elevation angle assumed to be zero.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the direction vector and *xy* plane. This angle is positive when measured towards the *z*-axis. See "Azimuth and Elevation Angles".

Example: `[45 60; 0 10]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `CoordinateSystem,'polar',Type,'directivity'`

PropagationSpeed — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

Weights — Array weights
1 (default) | N-by-1 complex-valued column vector | N-by-L complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an N-by-1 complex-valued column vector or N-by-L complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension N is the number of elements in the array. The dimension L is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-L row vector	Applies a set of weights for the single frequency or for all L frequencies.
N-by-L complex-valued matrix	1-by-L row vector	Applies each of the L columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

D — Directivity
M-by-L matrix

Directivity, returned as an M-by-L matrix. Each row corresponds to one of the M angles specified by `ANGLE`. Each column corresponds to one of the L frequency values specified in `FREQ`. Directivity units are in dBi where dBi is defined as the gain of an element relative to an isotropic radiator.

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

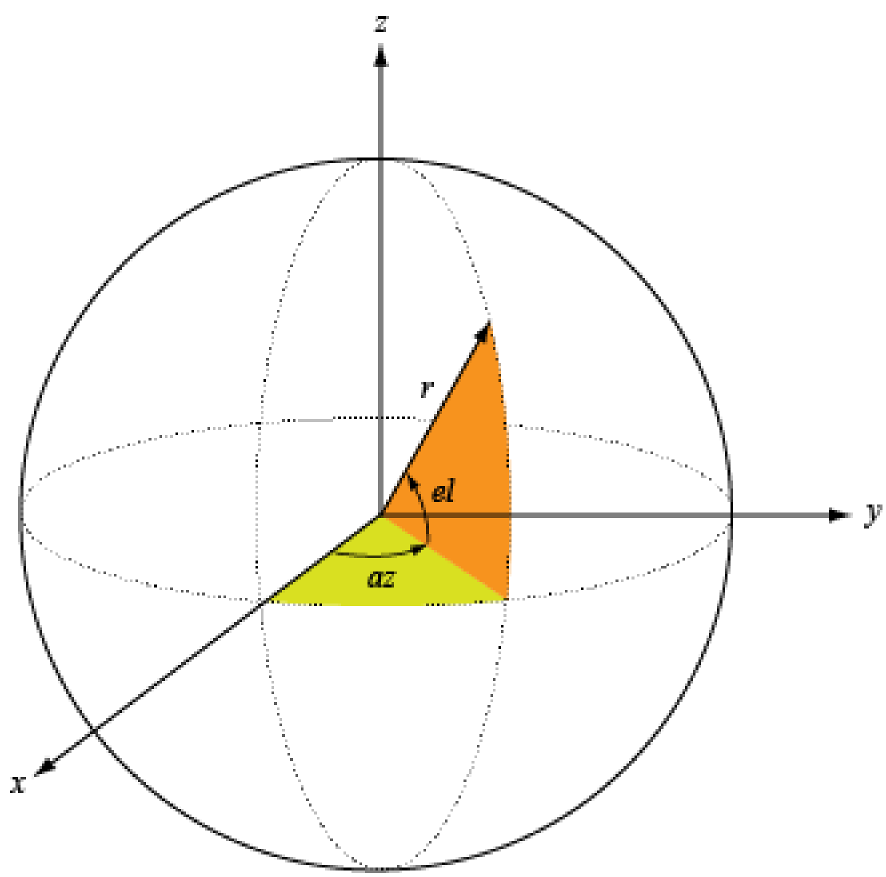
$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{rad}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Azimuth and Elevation Angles

Define the azimuth and elevation conventions used in the toolbox.

The azimuth angle of a vector is the angle between the *x*-axis and its orthogonal projection onto the *xy*-plane. The angle is positive when going from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180° and 180° degrees, inclusive. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy*-plane. Elevation angles lie between –90° and 90° degrees, inclusive.



Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`pattern` | `patternAzimuth` | `patternElevation`

getElementNormal

Normal vectors for array elements

Syntax

```
normvec = getElementNormal(array)
normvec = getElementNormal(array,elemidx)
```

Description

`normvec = getElementNormal(array)` returns the normal vectors `normvec` to the `array`.

`normvec = getElementNormal(array,elemidx)` returns only the normals of the elements that are specified in the element index vector `elemidx`.

Examples

ULA Element Normals

Construct three ULAs with elements along the *x*-, *y*-, and *z*-axes. Obtain the element normals.

First, choose the array axis along the *x*-axis.

```
sULA1 = phased.ULA('NumElements',5,'ArrayAxis','x');
norm = getElementNormal(sULA1)
```

```
norm = 2×5

    90    90    90    90    90
     0     0     0     0     0
```

The element normal vectors point along the *y*-axis.

Next, choose the array axis along the *y*-axis.

```
sULA2 = phased.ULA('NumElements',5,'ArrayAxis','y');
norm = getElementNormal(sULA2)
```

```
norm = 2×5

     0     0     0     0     0
     0     0     0     0     0
```

The element normal vectors point along the *x*-axis.

Finally, set the array axis along the *z*-axis. Obtain the normal vectors of the odd-numbered elements.

```
sULA3 = phased.ULA('NumElements',5,'ArrayAxis','z');
norm = getElementNormal(sULA3,[1,3,5])
```

```
norm = 2×3

     0     0     0
     0     0     0
```

The element normal vectors also point along the *x*-axis.

Input Arguments

`array` — Phased array System object
System object

Phased array, specified as a System object.

`elemidx` — Element index vector
all element indexes (default) | vector of positive integers

Element index vector, specified as a vector of positive integers each of which takes a value from 1 to *N*. The dimension *N* is the number of elements of the array.

Example: `[1,2,3]`

Output Arguments

`normvec` — Normal vector
2-by-*M* real-valued matrix

Normal vector of array elements, returned as a 2-by-*M* real matrix. Each column of `normvec` specifies the normal direction of the corresponding element in the local coordinate system in the form `[azimuth;elevation]`. If the input argument `elemidx` is not specified, *M* is the number of elements of the array, *N*. If `elemidx` is specified, *M* is the size of `elemidx`. The azimuth angle is defined as the angle from *x*-axis toward *y* axis. The elevation angle is defined as the angle from *xy*-plane toward *z*-axis. Units are in degrees.

For uniform linear arrays, the normal vector depends upon the value of the `ArrayAxis` property.

ArrayAxis Property Value	Array Normal Direction
`'x'`	azimuth = 90°, elevation = 0° (*y*-axis)
`'y'`	azimuth = 0°, elevation = 0° (*x*-axis)
`'z'`	azimuth = 0°, elevation = 0° (*x*-axis)

- For a conformal array, the origin of the local coordinate system, i.e., the phase center of the array, is defined at an arbitrary point. In local coordinate system, x, y, and z axes are three orthogonal axes with x axis defines the normal direction of the conformal array. The elements are located in the 3D space.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

getElementPosition

Positions of array elements

Syntax

```
pos = getElementPosition(array)
pos = getElementPosition(array,elemidx)
```

Description

`pos = getElementPosition(array)` returns the element positions `pos` of the System object `array`.

`pos = getElementPosition(array,elemidx)` returns only the positions of the elements that are specified in the element index vector `elemidx`.

Examples

ULA Element Positions

Construct a ULA with 5 elements along the z-axis. Obtain the element positions.

```
sULA = phased.ULA('NumElements',5,'ArrayAxis','z');
pos = getElementPosition(sULA)
```

```
pos = 3×5

         0         0         0         0         0
         0         0         0         0         0
   -1.0000   -0.5000         0    0.5000    1.0000
```

Input Arguments

`array` — Phased array System object
System object

Phased array, specified as a System object.

`elemidx` — Element index vector
all element indexes (default) | vector of positive integers

Element index vector, specified as a vector of positive integers each of which takes a value from 1 to *N*. The quantity *N* is the number of elements of the array.

Example: `[1,2,3]`

Output Arguments

pos — Positions of array elements
3-by-*M* real matrix

Positions of array elements, returned as a 3-by-*M* real matrix. Each column of `pos` defines the position of an element in the local coordinate system, using the form `[x;y;z]`. If the input argument `elemidx` is not specified, *M* is the number of elements of the array, *N*. If `elemidx` is specified, *M* is the dimension of `elemidx`. Units are in meters

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

getNumElements

Number of elements in an array

Syntax

```
N = getNumElements(array)
```

Description

`N = getNumElements(array)` returns the number of elements `N` in the array `array`.

Examples

Number of Elements of UCA

Create a UCA with the default number of elements. Verify that there are five elements.

```
sArray = phased.UCA();
N = getNumElements(sArray)
```

```
N = 5
```

Input Arguments

`array` — Phased array System object
System object

Phased array, specified as a System object.

Output Arguments

`N` — Number of array elements
positive integer

Number of elements of array, returned as a positive integer.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

getTaper

Array element tapers

Syntax

```
wts = getTaper(array)
```

Description

`wts = getTaper(array)` returns the tapers `wts` applied to each element of the phased array `array`. Tapers are often referred to as weights.

Examples

Construct ULA with Taylor Window

Construct a 5-element ULA with a Taylor window taper. Then, obtain the element taper values.

```
taper = taylorwin(5)';
array = phased.ULA(5,'Taper',taper);
w = getTaper(array)
```

```
w = 5×1

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

Input Arguments

`array` — Phased array System object
System object

Phased array, specified as a Phased Array System Toolbox System object.

Output Arguments

`wts` — Array element tapers
N-by-1 complex-valued vector

Array element tapers, returned as an *N*-by-1 complex-valued vector, where *N* is the number of elements in the array.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

isPolarizationCapable

Array polarization capability

Syntax

`capflag = isPolarizationCapable(array)`

Description

`capflag = isPolarizationCapable(array)` returns a Boolean value, `capflag`, indicating whether the `array` supports polarization. An array supports polarization if it can create or respond to polarized fields.

Examples

Short-Dipole Antenna ULA Supports Polarization

Show that an array of `phased.ShortDipoleAntennaElement` antenna elements supports polarization.

```
antenna = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[1e9 10e9]);
array = phased.ULA('NumElements',3,'Element',antenna);
isPolarizationCapable(array)
```

```
ans = logical
   1
```

The returned value of `1` shows that this array supports polarization.

Input Arguments

`array` — Array
Phased Array System Toolbox System object

Array, specified as a Phased Array System Toolbox System object.

Output Arguments

`capflag` — Polarization-capability flag
`true` | `false`

Polarization-capability flag returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

pattern

Package: phased

Plot array directivity and patterns

Syntax

```
pattern(array,FREQ)
pattern(array,FREQ,AZ)
pattern(array,FREQ,AZ,EL)
pattern( ___ ,Name,Value)
[PAT,AZ_ANG,EL_ANG] = pattern( ___ )
```

Description

pattern(array,FREQ) plots the 3-D array directivity pattern (in dBi) for the array specified in array. The operating frequency is specified in FREQ. You can use this function to display the patterns of arrays that support polarization.

pattern(array,FREQ,AZ) plots the array directivity pattern at the specified azimuth angle.

pattern(array,FREQ,AZ,EL) plots the array directivity pattern at specified azimuth and elevation angles.

pattern(___ ,Name,Value) plots the array pattern with additional options specified by one or more Name,Value pair arguments.

[PAT,AZ_ANG,EL_ANG] = pattern(___) returns the array pattern in PAT. The AZ_ANG output contains the coordinate values corresponding to the rows of PAT. The EL_ANG output contains the coordinate values corresponding to the columns of PAT. If the 'CoordinateSystem' parameter is set to 'uv', then AZ_ANG contains the *U* coordinates of the pattern and EL_ANG contains the *V* coordinates of the pattern. Otherwise, they are in angular units in degrees. *UV* units are dimensionless.

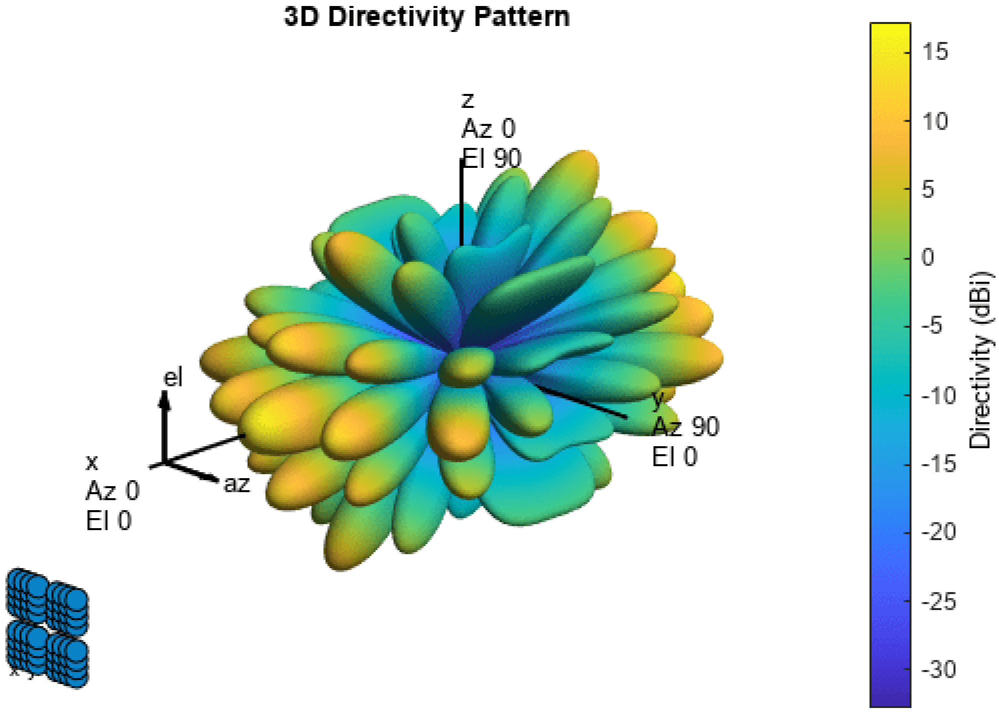
Examples

Plot Response of NR Rectangular Panel Array

Construct a 5G antenna array where the grid is 2-by-2 and each panel is a 4-by-4 array. Each antenna element consists of two short-dipole antennas with different dipole axis directions. The antenna elements are spaced 1/2 wavelength apart and the panels are spaced 3 wavelengths apart. Plot the response pattern of the array assuming an operating frequency of 6 GHz.

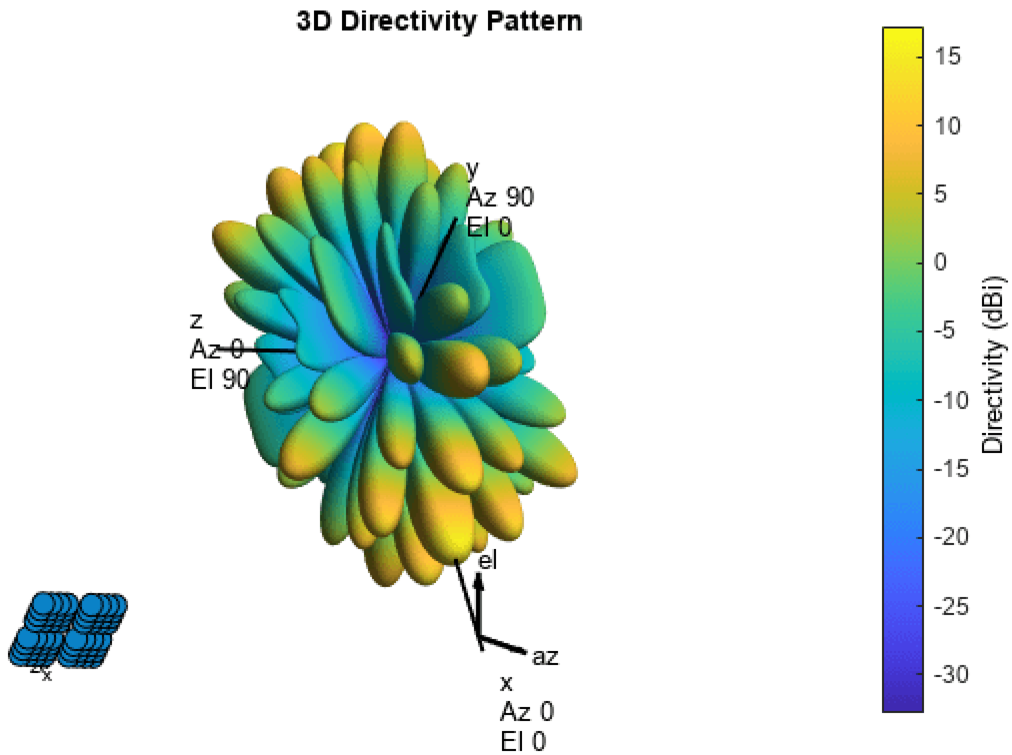
```
c = physconst('LightSpeed');
fc = 6e9;
lambda = c/fc;
antenna1 = phased.ShortDipoleAntennaElement('AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('AxisDirection','X');
array = phased.NRRectangularPanelArray('ElementSet', ...
```

```
        {antenna1, antenna2},'Size',[4, 4, 2, 2],'Spacing', ...
        [0.5*lambda, 0.5*lambda,3*lambda, 3*lambda]);
pattern(array,fc,'ShowArray',true)
```



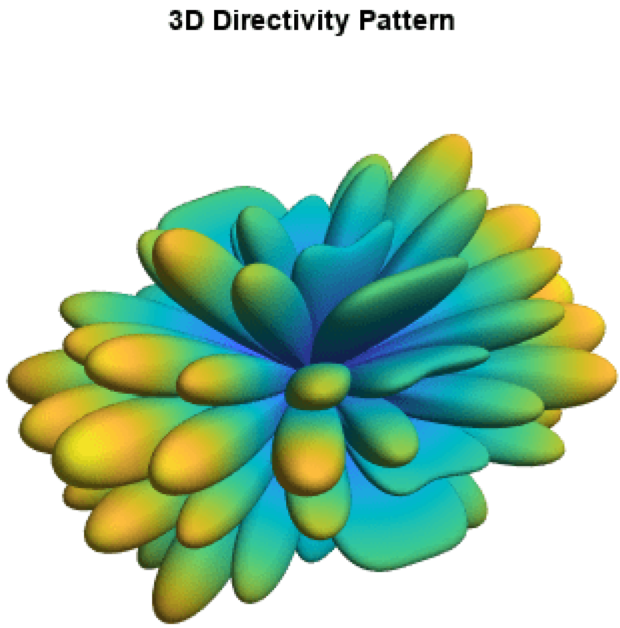
Use the `Orientation` property of `pattern` to change the orientation 80˚ along the *x*-axis, 30˚ along the *y*-axis and 60˚ along the *z*-axis.

```
pattern(array,fc,'Orientation',[80;30;60],'ShowArray',true)
```



Disable the display of local coordinates and the colorbar.

```
pattern(array,fc,'ShowLocalCoordinate',false,'ShowColorBar',false)
```



Input Arguments

array — Phased array

Phased Array System Toolbox System object

Phased array, specified as a Phased Array System Toolbox System object.

FREQ — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element. Otherwise, the element produces no response and the directivity is returned as `–Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.
- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

AZ — Azimuth angles
`[-180:180]` (default) | 1-by-*N* real-valued row vector

Azimuth angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of azimuth angles. Angle units are in degrees. Azimuth angles must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. When measured from the *x*-axis toward the *y*-axis, this angle is positive.

Example: `[-45:2:45]`

Data Types: `double`

EL — Elevation angles
`[-90:90]` (default) | 1-by-*M* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of desired elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured towards the *z*-axis.

Example: `[-75:1:70]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `CoordinateSystem,'polar',Type,'directivity'`

`CoordinateSystem` — Plotting coordinate system
`'polar'` (default) | `'rectangular'` | `'uv'`

Plotting coordinate system of the pattern, specified as the comma-separated pair consisting of `'CoordinateSystem'` and one of `'polar'`, `'rectangular'`, or `'uv'`. When `'CoordinateSystem'` is set to `'polar'` or `'rectangular'`, the `AZ` and `EL` arguments specify the pattern azimuth and elevation, respectively. `AZ` values must lie between –180° and 180°. `EL` values must lie between –90° and 90°. If `'CoordinateSystem'` is set to `'uv'`, `AZ` and `EL` then specify *U* and *V* coordinates, respectively. `AZ` and `EL` must lie between -1 and 1.

Example: `'uv'`

Data Types: `char`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- 'directivity' — directivity pattern measured in dBi.
- 'efield' — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- 'power' — power pattern of the sensor or array defined as the square of the field pattern.
- 'powerdb' — power pattern converted to dB.

Example: 'powerdb'

Data Types: char

Orientation — Array orientation
[0;0;0]. (default) | 3-by-1 real-valued column vector

Array orientation, specified as a 3-by-1 real-valued column vector containing the rotation angles with respect to the *x*-, *y*-, and *z*-axes of the local coordinate system, respectively.

Normalize — Display normalize pattern
true (default) | false

Display normalized pattern, specified as the comma-separated pair consisting of 'Normalize' and a Boolean. Set this parameter to true to display a normalized pattern. This parameter does not apply when you set 'Type' to 'directivity'. Directivity patterns are already normalized.

Data Types: logical

ShowArray — View array geometry
false (default) | true

View the array geometry along with the 3D radiation pattern, specified as false or true.

Data Types: logical

ShowLocalCoordinates — Show local coordinate axes
true (default) | false

Show the local coordinate axes, specified as true or false.

Data Types: logical

ShowColorbar — Show colorbar
true (default) | false

Show the colorbar, specified as true or false.

Data Types: logical

Parent — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

PlotStyle — Plotting style
'overlay' (default) | 'waterfall'

Plotting style, specified as the comma-separated pair consisting of 'Plotstyle' and either 'overlay' or 'waterfall'. This parameter applies when you specify multiple frequencies in FREQ in 2-D plots. You can draw 2-D plots by setting one of the arguments AZ or EL to a scalar.

Data Types: `char`

`Polarization` — Polarization type
`'combined'` (default) | `'H'` | `'V'`

Polarization type, specified as the comma-separated pair consisting of `'Polarization'` and either `'combined'`, `'H'`, or `'V'`. If `Polarization` is `'combined'`, the horizontal and vertical polarization patterns are combined. If `Polarization` is `'H'`, only the horizontal polarization is displayed. If `Polarization` is `'V'`, only the vertical polarization is displayed.

Dependencies

To enable this property, set the `array` argument to an array that supports polarization and then set the `'Type'` name-value pair to `'efield'`, `'power'`, or `'powerdb'`.

Data Types: `char` | `string`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`

Complex Number Support: Yes

Output Arguments

PAT — array pattern
N-by-*M* real-valued matrix

Array pattern, returned as an *N*-by-*M* real-valued matrix. The pattern is a function of azimuth and elevation. The rows of `PAT` correspond to the azimuth angles in the vector specified by `EL_ANG`. The columns correspond to the elevation angles in the vector specified by `AZ_ANG`.

AZ_ANG — Azimuth angles
scalar | 1-by-*N* real-valued row vector

Azimuth angles for displaying directivity or response pattern, returned as a scalar or 1-by-*N* real-valued row vector corresponding to the dimension set in `AZ`. The columns of `PAT` correspond to the values in `AZ_ANG`. Units are in degrees.

EL_ANG — Elevation angles
scalar | 1-by-*M* real-valued row vector

Elevation angles for displaying directivity or response, returned as a scalar or 1-by-*M* real-valued row vector corresponding to the dimension set in `EL`. The rows of `PAT` correspond to the values in `EL_ANG`. Units are in degrees.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

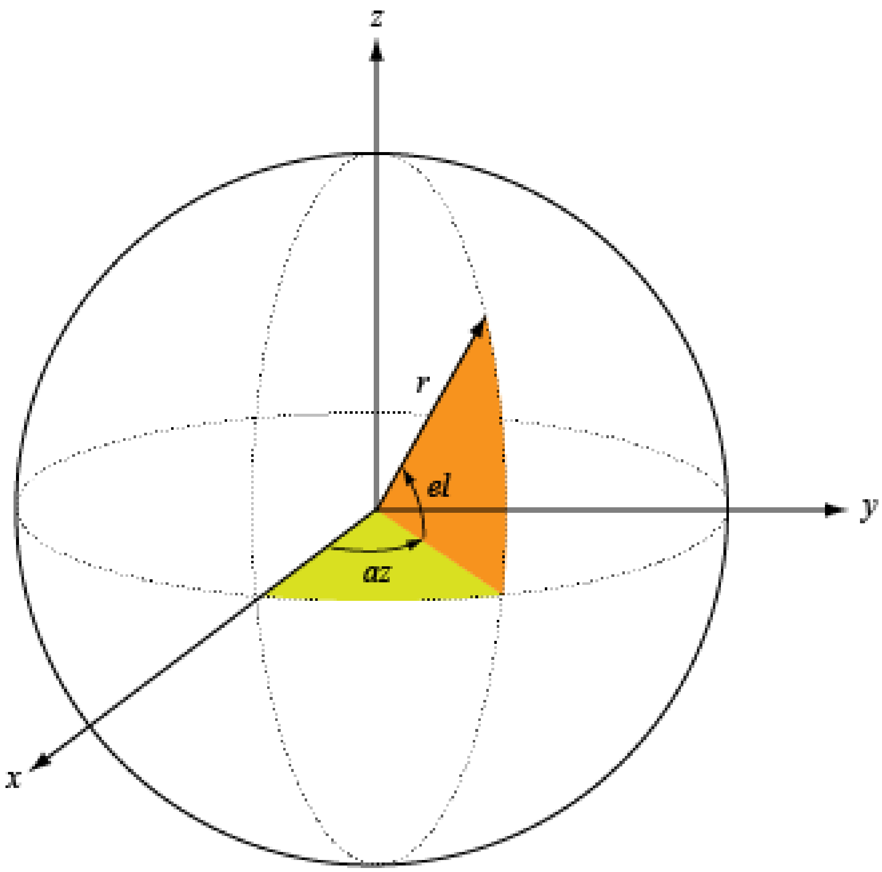
$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Azimuth and Elevation Angles

Define the azimuth and elevation conventions used in the toolbox.

The azimuth angle of a vector is the angle between the *x*-axis and its orthogonal projection onto the *xy*-plane. The angle is positive when going from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180° and 180° degrees, inclusive. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy*-plane. Elevation angles lie between –90° and 90° degrees, inclusive.



Version History

Introduced in R2021a

patternAzimuth

Package: `phased`

Plot array directivity or pattern versus azimuth

Syntax

```
patternAzimuth(array,FREQ)
patternAzimuth(array,FREQ,EL)
patternAzimuth(array,FREQ,EL)
patternAzimuth(array,FREQ,EL,Name,Value)
PAT = patternAzimuth( ___ )
```

Description

`patternAzimuth(array,FREQ)` plots the 2-D array directivity pattern versus azimuth (in dBi) for the `array` at zero degrees elevation angle. The argument `FREQ` specifies the operating frequency.

The integration used when computing array directivity has a minimum sampling grid of 0.1 degrees. If an array pattern has a beamwidth smaller than this, the directivity value will be inaccurate.

`patternAzimuth(array,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the `array` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(array,FREQ,EL)`, in addition, plots the 2-D array directivity pattern versus azimuth (in dBi) for the `array` at the elevation angle specified by `EL`. When `EL` is a vector, multiple overlaid plots are created.

`patternAzimuth(array,FREQ,EL,Name,Value)` plots the array pattern with additional options specified by one or more `Name,Value` pair arguments.

`PAT = patternAzimuth( ___ )` returns the array pattern. `PAT` is a matrix whose entries represent the pattern at corresponding sampling points specified by the `'Azimuth'` parameter and the `EL` input arguments.

Input Arguments

`array` — Phased array

Phased Array System Toolbox System object

Phased array, specified as a Phased Array System Toolbox System object.

`FREQ` — Frequency for computing directivity and patterns

positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, `FREQ` must lie within the range of values specified by the `FrequencyRange` or `FrequencyVector` property of the element.

Otherwise, the element produces no response and the directivity is returned as –`Inf`. Most elements use the `FrequencyRange` property except for `phased.CustomAntennaElement` and `phased.CustomMicrophoneElement`, which use the `FrequencyVector` property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –`Inf`.

Example: `[1e8 2e6]`

Data Types: `double`

EL — Elevation angles
1-by-*N* real-valued row vector

Elevation angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector. The quantity *N* is the number of requested elevation directions. Angle units are in degrees. The elevation angle must lie between –90° and 90°.

The elevation angle is the angle between the direction vector and the *xy* plane. When measured toward the *z*-axis, this angle is positive.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `CoordinateSystem,'polar',Type,'directivity'`

Type — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

PropagationSpeed — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
1 (default) | *N*-by-1 complex-valued column vector | *N*-by-*L* complex-valued matrix

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an *N*-by-1 complex-valued column vector or *N*-by-*L* complex-valued matrix. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension *N* is the number of elements in the array. The dimension *L* is the number of frequencies specified by `FREQ`.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 complex-valued column vector	Scalar or 1-by-*L* row vector	Applies a set of weights for the single frequency or for all *L* frequencies.
N-by-*L* complex-valued matrix	1-by-*L* row vector	Applies each of the *L* columns of `'Weights'` for the corresponding frequency in `FREQ`.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(N,M)`

Data Types: `double`
Complex Number Support: Yes

`Azimuth` — Azimuth angles
`[-180:180]` (default) | 1-by-*P* real-valued row vector

Azimuth angles, specified as the comma-separated pair consisting of `'Azimuth'` and a 1-by-*P* real-valued row vector. Azimuth angles define where the array pattern is calculated.

Example: `'Azimuth',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

PAT — Array directivity or pattern
L-by-*N* real-valued matrix

Array directivity or pattern, returned as an *L*-by-*N* real-valued matrix. The dimension *L* is the number of azimuth values determined by the `'Azimuth'` name-value pair argument. The dimension *N* is the number of elevation angles, as determined by the `EL` input argument.

More About

Directivity (dBi)

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

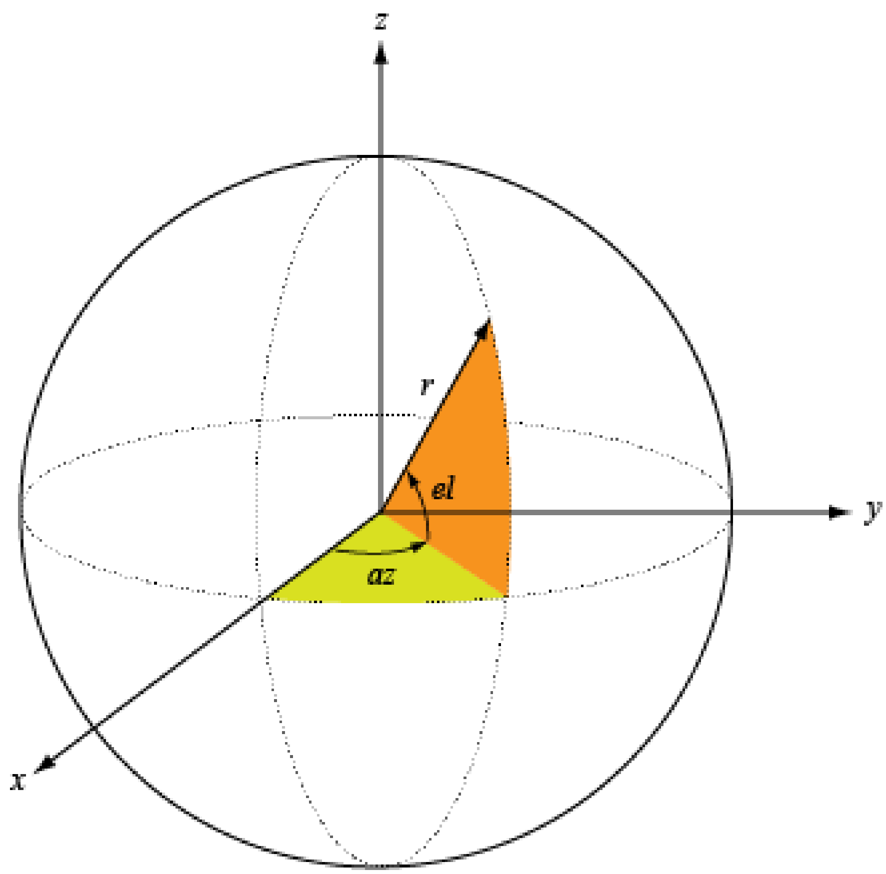
$$D = 4\pi\frac{U_{\mathrm{rad}}(\theta,\varphi)}{P_{\mathrm{total}}}$$

where $U_{\mathrm{rad}}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Azimuth and Elevation Angles

Define the azimuth and elevation conventions used in the toolbox.

The azimuth angle of a vector is the angle between the *x*-axis and its orthogonal projection onto the *xy*-plane. The angle is positive when going from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180° and 180° degrees, inclusive. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy*-plane. Elevation angles lie between –90° and 90° degrees, inclusive.



Version History

Introduced in R2021a

patternElevation

Package: phased

Plot array directivity or pattern versus elevation

Syntax

```
patternElevation(array,FREQ)
patternElevation(array,FREQ,AZ)
patternElevation(array,FREQ,AZ,Name,Value)
PAT = patternElevation( ___ )
```

Description

patternElevation(array,FREQ) plots the 2-D array directivity pattern versus elevation (in dBi) for the array at zero degrees azimuth angle. When AZ is a vector, multiple overlaid plots are created. The argument FREQ specifies the operating frequency.

patternElevation(array,FREQ,AZ), in addition, plots the 2-D element directivity pattern versus elevation (in dBi) at the azimuth angle specified by AZ. When AZ is a vector, multiple overlaid plots are created.

patternElevation(array,FREQ,AZ,Name,Value) plots the array pattern with additional options specified by one or more Name,Value pair arguments.

PAT = patternElevation(___) returns the array pattern. PAT is a matrix whose entries represent the pattern at corresponding sampling points specified by the 'Elevation' parameter and the AZ input argument.

Input Arguments

array — phased array

System object

Phased array, specified as a System object.

Example: array = phased.UCA;

FREQ — Frequency for computing directivity and pattern

positive scalar

Frequency for computing directivity and pattern, specified as a positive scalar. Frequency units are in hertz.

- For an antenna or microphone element, FREQ must lie within the range of values specified by the FrequencyRange or the FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.

- For an array of elements, `FREQ` must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as `–Inf`.

Example: `1e8`

Data Types: `double`

`AZ` — Azimuth angles for computing directivity and pattern
1-by-*N* real-valued row vector

Azimuth angles for computing sensor or array directivities and patterns, specified as a 1-by-*N* real-valued row vector where *N* is the number of desired azimuth directions. Angle units are in degrees. The azimuth angle must lie between –180° and 180°.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy* plane. This angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `[0,10,20]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `CoordinateSystem,'polar',Type,'directivity'`

`Type` — Displayed pattern type
`'directivity'` (default) | `'efield'` | `'power'` | `'powerdb'`

Displayed pattern type, specified as the comma-separated pair consisting of `'Type'` and one of

- `'directivity'` — directivity pattern measured in dBi.
- `'efield'` — field pattern of the sensor or array. For acoustic sensors, the displayed pattern is for the scalar sound field.
- `'power'` — power pattern of the sensor or array defined as the square of the field pattern.
- `'powerdb'` — power pattern converted to dB.

Example: `'powerdb'`

Data Types: `char`

`PropagationSpeed` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as the comma-separated pair consisting of `'PropagationSpeed'` and a positive scalar in meters per second.

Example: `'PropagationSpeed',physconst('LightSpeed')`

Data Types: `double`

`Weights` — Array weights
M-by-1 complex-valued column vector

Array weights, specified as the comma-separated pair consisting of `'Weights'` and an M-by-1 complex-valued column vector. Array weights are applied to the elements of the array to produce array steering, tapering, or both. The dimension M is the number of elements in the array.

Note Use complex weights to steer the array response toward different directions. You can create weights using the `phased.SteeringVector` System object or you can compute your own weights. In general, you apply Hermitian conjugation before using weights in any Phased Array System Toolbox function or System object such as `phased.Radiator` or `phased.Collector`. However, for the `directivity`, `pattern`, `patternAzimuth`, and `patternElevation` methods of any array System object use the steering vector without conjugation.

Example: `'Weights',ones(10,1)`

Data Types: `double`
Complex Number Support: Yes

`Elevation` — Elevation angles
`[-90:90]` (default) | 1-by-P real-valued row vector

Elevation angles, specified as the comma-separated pair consisting of `'Elevation'` and a 1-by-P real-valued row vector. Elevation angles define where the array pattern is calculated.

Example: `'Elevation',[-90:2:90]`

Data Types: `double`

`Parent` — Handle to axis
scalar

Handle to the axes along which the array geometry is displayed specified as a scalar.

Output Arguments

`PAT` — Array directivity or pattern
L-by-N real-valued matrix

Array directivity or pattern, returned as an L-by-N real-valued matrix. The dimension L is the number of elevation angles determined by the `'Elevation'` name-value pair argument. The dimension N is the number of azimuth angles determined by the `AZ` argument.

More About

Directivity

Directivity describes the directionality of the radiation pattern of a sensor element or array of sensor elements.

Higher directivity is desired when you want to transmit more radiation in a specific direction. Directivity is the ratio of the transmitted radiant intensity in a specified direction to the radiant intensity transmitted by an isotropic radiator with the same total transmitted power

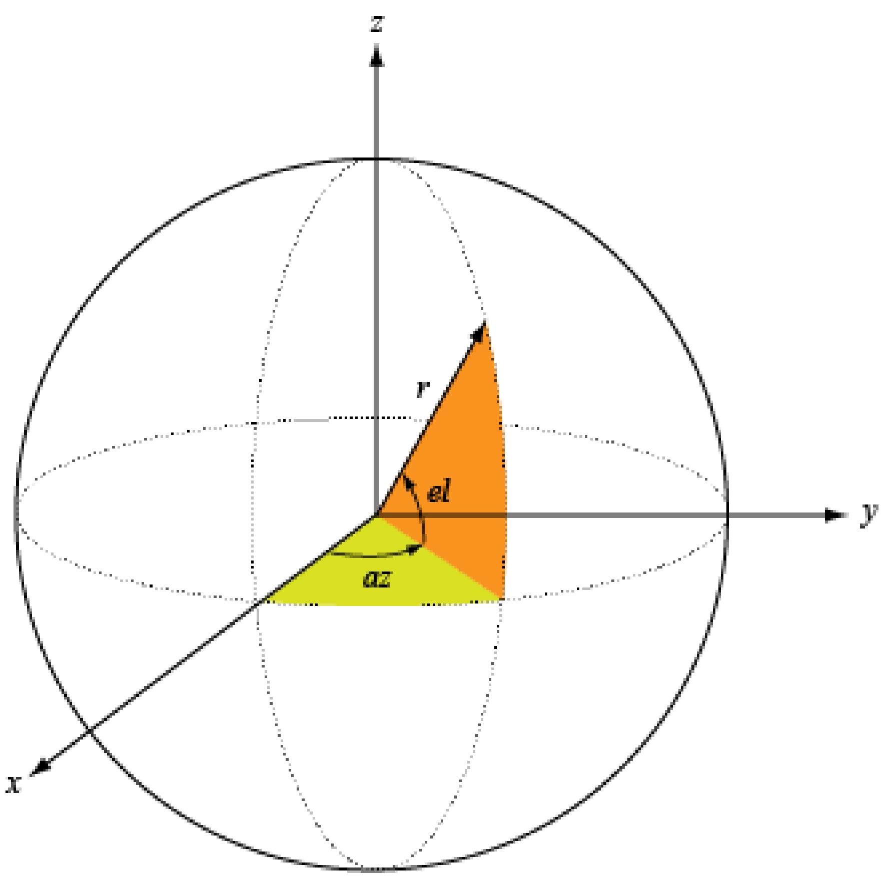
$$D = 4\pi \frac{U_{\mathrm{rad}}(\theta, \varphi)}{P_{\mathrm{total}}}$$

where $U_{rad}(\theta,\varphi)$ is the radiant intensity of a transmitter in the direction (θ,φ) and P_{total} is the total power transmitted by an isotropic radiator. For a receiving element or array, directivity measures the sensitivity toward radiation arriving from a specific direction. The principle of reciprocity shows that the directivity of an element or array used for reception equals the directivity of the same element or array used for transmission. When converted to decibels, the directivity is denoted as *dBi*. For information on directivity, read the notes on “Element Directivity” and “Array Directivity”.

Azimuth and Elevation Angles

Define the azimuth and elevation conventions used in the toolbox.

The azimuth angle of a vector is the angle between the *x*-axis and its orthogonal projection onto the *xy*-plane. The angle is positive when going from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180° and 180° degrees, inclusive. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy*-plane. Elevation angles lie between –90° and 90° degrees, inclusive.



Version History

Introduced in R2021a

perturbations

Perturbations defined on array

Syntax

```
perts = perturbations(array)
perts = perturbations(array,prop)
perts = perturbations(array,prop,'None')
perts = perturbations(array,prop,'Normal',mean,sigma)
perts = perturbations(array,prop,'Uniform',minval,maxval)
perts = perturbations(array,prop,'RandomFail',failprob)
```

Description

`perts = perturbations(array)` returns a table of all allowed perturbations `perts` defined for the `array`. This table contains a list of all properties that can be perturbed, the probability type of applied perturbation, and the parameters of the probability type.

`perts = perturbations(array,prop)` lets you view the current perturbation defined for the array property `prop`.

`perts = perturbations(array,prop,'None')` specifies that the property `prop` is not perturbed.

`perts = perturbations(array,prop,'Normal',mean,sigma)` specifies that the perturbation is drawn from a normal probability distribution defined by its `mean` and standard deviation `sigma`. To use this syntax, set `prop` to `'ElementPositions'`, `'TaperMagnitude'`, or `'TaperPhase'`.

`perts = perturbations(array,prop,'Uniform',minval,maxval)` specifies that the perturbation is drawn from a uniform probability distribution with a range defined by the interval [`minval`,`maxval`]. To use this syntax, set `prop` to `'ElementPositions'`, `'TaperMagnitude'`, or `'TaperPhase'`.

`perts = perturbations(array,prop,'RandomFail',failprob)` specifies that the perturbation is a mask indicating whether an element is functioning based on the element fail property `failprob`. To use this syntax, set prop to `'ElementFailure'`.

Examples

Perturbations of Uniform Rectangular Array

Create an 8-by-3 uniform rectangular array (URA). The array operates at 300 MHz and its elements are spaced one-half wavelength apart.

```
freq = 300.0e6;
lambda = physconst('Lightspeed')/freq;
d = lambda/2;
array = phased.URA(8,3,ElementSpacing=[d,d]);
```

Initially, there are no perturbations to the array.

```
perts = perturbations(array)
```

```
perts=4×3 table
         Property            Type             Value      
    ___________________    ________    __________________

    {'ElementPosition'}    {'None'}    {[NaN]}    {[NaN]}
    {'TaperMagnitude' }    {'None'}    {[NaN]}    {[NaN]}
    {'TaperPhase'     }    {'None'}    {[NaN]}    {[NaN]}
    {'ElementFailure' }    {'None'}    {[NaN]}    {[NaN]}
```

Randomly perturb the element positions according to a normal distribution. Use a position variance of 16th of a wavelength.

```
perts = perturbations(array,'ElementPosition','Normal',0,lambda/16)
```

```
perts=4×3 table
         Property             Type              Value        
    ___________________    __________    ____________________

    {'ElementPosition'}    {'Normal'}    {[   0]}    {[0.0625]}
    {'TaperMagnitude' }    {'None'  }    {[NaN]}    {[   NaN]}
    {'TaperPhase'     }    {'None'  }    {[NaN]}    {[   NaN]}
    {'ElementFailure' }    {'None'  }    {[NaN]}    {[   NaN]}
```

Then perturb the magnitude of the element weights according to a normal distribution with a mean value of 0.1 and a variance of 0.02.

```
perts = perturbations(array,'TaperMagnitude','Normal',0.1,0.02)
```

```
perts=4×3 table
         Property             Type               Value          
    ___________________    __________    ________________________

    {'ElementPosition'}    {'Normal'}    {[     0]}    {[0.0625]}
    {'TaperMagnitude' }    {'Normal'}    {[0.1000]}    {[0.0200]}
    {'TaperPhase'     }    {'None'  }    {[   NaN]}    {[   NaN]}
    {'ElementFailure' }    {'None'  }    {[   NaN]}    {[   NaN]}
```

Perturb the phase of the element weights according to a uniform distribution between −40 and 40 degrees.

```
perts = perturbations(array,'TaperPhase','Uniform',-40,40)
```

```
perts=4×3 table
         Property             Type                Value          
    ___________________    ___________    ________________________

    {'ElementPosition'}    {'Normal' }    {[     0]}    {[0.0625]}
    {'TaperMagnitude' }    {'Normal' }    {[0.1000]}    {[0.0200]}
    {'TaperPhase'     }    {'Uniform'}    {[   -40]}    {[    40]}
    {'ElementFailure' }    {'None'   }    {[   NaN]}    {[   NaN]}
```

Set a 20% percent failure rate for the elements.

```
perts = perturbations(array,'ElementFailure','RandomFail',0.2)
```

```
perts=4×3 table
         Property              Type                 Value
    ___________________    ______________    ________________________

    {'ElementPosition'}    {'Normal'    }    {[     0]}    {[0.0625]}
    {'TaperMagnitude' }    {'Normal'    }    {[0.1000]}    {[0.0200]}
    {'TaperPhase'     }    {'Uniform'   }    {[   -40]}    {[    40]}
    {'ElementFailure' }    {'RandomFail'}    {[0.2000]}    {[   NaN]}
```

Input Arguments

array — Phased array

Phased Array System Toolbox System object

Phased array, specified as a Phased Array System Toolbox System object.

prop — Perturbed array property

'ElementPosition' | 'TaperMagnitude' | 'TaperPhased' | 'ElementFailure'

Perturbed property of array, specified as 'ElementPosition', 'TaperMagnitude', 'TaperPhased', or 'ElementFailure'.

Example: 'TaperPhased'

Data Types: string

mean — Mean value of normal distribution

scalar

Mean value of normal distribution, specified as a scalar. Units depend on the property prop.

'ElementPosition'	meters
'TaperMagnitude'	dimensionless
'TaperPhased'	radians

Example: 12

Dependencies

To enable this argument, set the perturbed array property prop to 'ElementPosition', 'TaperMagnitude', or 'TaperPhased' and the perturbation type to 'Normal'.

Data Types: double

sigma — Standard deviation of normal distribution

positive scalar

Standard deviation of normal distribution, specified as a positive scalar. Units depend on the property prop.

'ElementPosition'	meters

'TaperMagnitude'	dimensionless
'TaperPhased'	radians

Example: 1.0

Dependencies

To enable this argument, set the perturbed array property `prop` to `'ElementPosition'`, `'TaperMagnitude'`, `or` `'TaperPhased'` and the perturbation `type` to `'Normal'`.

Data Types: `double`

`minval` — Minimum value of range of uniform probability distribution
scalar

Minimum value of range of uniform probability distribution, specified as a scalar. When applied to the `'TaperPhase'` property. the difference between `minval` and `maxval` should be less than or equal to 2π. Units depend on the property `prop`.

'ElementPosition'	meters
'TaperMagnitude'	dimensionless
'TaperPhased'	radians

Example: `0`

Dependencies

To enable this argument, set the perturbed array property `prop` to `'ElementPosition'`, `'TaperMagnitude'`, `or` `'TaperPhased'` and the perturbation `type` to `'Uniform'`.

Data Types: `double`

`maxval` — Maximum value of range of uniform probability distribution
scalar

Maximum value of range of uniform probability distribution

Example: `1`

Dependencies

Maximum value of range of uniform probability distribution, specified as a scalar. When applied to the `'TaperPhase'` property. the difference between `minval` and `maxval` should be less than or equal to 2π. Units depend on the property `prop`.

'ElementPosition'	meters
'TaperMagnitude'	dimensionless
'TaperPhased'	radians

Data Types: `double`

`failprob` — Probability of failure
positive scalar

Probability of failure, specified as a non-negative scalar greater than or equal to zero and less than one. Zero means that the elements will never fail. Otherwise, there is a some probability of failure.

Example: `0.01`

Dependencies

To enable this argument, set the array property `prop` to `'ElementFailure'` and the perturbation type to `'RandomFail'`.

Data Types: `double`

Output Arguments

`perts` — List of possible perturbations
MATLAB table

List of possible perturbations, returned as a MATLAB `table`. See “Perturbed properties and perturbation types” on page 1-2209 for a list of perturbations properties and types.

Data Types: `table`

More About

Perturbed properties and perturbation types

You can perturb the array by selecting one of the properties to be perturbed and then applying the type of perturbation. Each type of perturbation has specific arguments.

Property	*Perturbation Type*	*Arguments*
`'ElementPosition'`	`'None'`	-
	`'Normal'`	`mean`, `sigma`
	`'Uniform'`	`minval`, `maxval`
`'TaperMagnitude'`	`'None'`	-
	`'Normal'`	`mean`, `sigma`
	`'Uniform'`	`minval`, `maxval`
`'TaperPhase'`	`'None'`	-
	`'Normal'`	`mean`, `sigma`
	`'Uniform'`	`minval`, `maxval`
`'ElementFailure'`	`'None'`	-
	`'RandomFail'`	`probfail`

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.ConformalArray` | `perturbedPattern` | `perturbedArray`

perturbedArray

Apply perturbations to phased array

Syntax

```
arrayp = perturbedArray(array)
[arrayp,offsets] = perturbedArray(array)
```

Description

`arrayp = perturbedArray(array)` creates a perturbed array `arrayp` from an original `array`. Use the `perturbations` function to apply one or perturbations to the `array`. Then create the perturbed array `arrayp` using the `perturbedArray` object function. The array perturbation are described in the perturbation table created by the `perturbations` object function. You can call this function many times to create a new perturbed array.

`[arrayp,offsets] = perturbedArray(array)` also returns the perturbation `offsets` used to create the perturbed array`arrayp`.

Examples

Show Perturbed Uniform Linear Array

Create an 11-element uniform linear array (ULA). The array operates at 300 MHz and its elements are spaced one-half wavelength apart. Perturb the element positions by 1/20th of a wavelength. A second perturbation sets an element failure rate of 10%.

```
freq = 300.0e6;
lambda = physconst('LightSpeed')/freq;
d = lambda/2;
array = phased.ULA(11,ElementSpacing=d/2);
```

Display the appled perturbations.

```
rng(2001)
perturbations(array,'ElementPosition','Normal',0,lambda/20);
perturbations(array,'ElementFailure','RandomFail',0.1)
```

```
ans=4×3 table
          Property               Type                   Value
    ____________________    _______________    __________________________

    {'ElementPosition'}    {'Normal'    }    {[     0]}    {[0.0500]}
    {'TaperMagnitude' }    {'None'      }    {[   NaN]}    {[   NaN]}
    {'TaperPhase'     }    {'None'      }    {[   NaN]}    {[   NaN]}
    {'ElementFailure' }    {'RandomFail'}    {[0.1000]}    {[   NaN]}
```

Create the perturbed array. Show that the array is a conformal array.

```
arrayp = perturbedArray(array)
```

```
arrayp = 
  phased.ConformalArray with properties:

            Element: [1x1 phased.IsotropicAntennaElement]
    ElementPosition: [3x11 double]
      ElementNormal: [2x11 double]
              Taper: [11x1 double]
```

Find the perturbations themselves using the `offsets` output.

```
[arrayp,offsets] = perturbedArray(array);
```

Display the element position perturbations.

```
offsets(1)
```

```
ans = struct with fields:
          Property: 'ElementPosition'
            Offset: [3x11 double]
    PerturbedValue: [3x11 double]
```

```
offsets(1).Offset
```

```
ans = 3×11

   -0.0372    0.0107    0.0092   -0.0774    0.0332   -0.1940   -0.0382    0.0454    0.0774    0.0
   -0.0177   -0.0745    0.1042    0.0210   -0.0359   -0.0391    0.0281   -0.0229    0.0281    0.0
    0.0048    0.0758    0.0528   -0.0372   -0.0066    0.0030    0.0323   -0.1047   -0.0887   -0.0
```

```
offsets(1).PerturbedValue
```

```
ans = 3×11

   -0.0372    0.0107    0.0092   -0.0774    0.0332   -0.1940   -0.0382    0.0454    0.0774    0.0
   -1.2668   -1.0738   -0.6452   -0.4786   -0.2858   -0.0391    0.2780    0.4768    0.7776    1.0
    0.0048    0.0758    0.0528   -0.0372   -0.0066    0.0030    0.0323   -0.1047   -0.0887   -0.0
```

Display the failed elements.

```
offsets(2)
```

```
ans = struct with fields:
          Property: 'ElementFailure'
            Offset: [11x1 double]
    PerturbedValue: [11x1 double]
```

```
offsets(2).Offset
```

```
ans = 11×1

     1
     1
     1
     1
     1
```

```
     1
     1
     0
     1
     1
     ⋮
```

```
offsets(2).PerturbedValue
```

```
ans = 11×1

     1
     1
     1
     1
     1
     1
     1
     0
     1
     1
     ⋮
```

Input Arguments

array — Phased array
Phased Array System Toolbox System object

Phased array, specified as a Phased Array System Toolbox System object.

Output Arguments

arrayp — Perturbed phased array
`phased.ConformalArray` | `phased.HeterogeneousConformalArray`

Phased array, returned as a `phased.ConformalArray` or a `phased.HeterogeneousConformalArray` System object. `arrayp` is a heterogeneous conformal array if the input unperturbed array is a any heterogeneous array.

offsets — Offsets used to define the perturbed array
array of `structs`

Perturbation offsets used to create the perturbed array, returned as an array of `structs`. Each `struct` describes one of the applied perturbations. The fields of each `struct` are:

`Property`	Perturbed property of array
`Offset`	Numeric array containing the offset or change in the property values.
`PerturbedValue`	Numeric array containing the newly perturbed property values.

Data Types: `struct`

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.ConformalArray` | `phased.HeterogeneousConformalArray` | `perturbations` | `perturbedPattern`

perturbedPattern

Display pattern of perturbed array

Syntax

```
pmc = perturbedPattern(array,freq)
pmc = perturbedPattern(array,freq,az)
pmc = perturbedPattern(array,freq,az,el)
pmc = perturbedPattern( ___ ,Name=Value)
[pmc,pnm,mpmc,varpmc] = perturbedPattern(array, ___ )
perturbedPattern(array, ___ )
```

Description

`pmc = perturbedPattern(array,freq)` returns a set of perturbed azimuth array patterns `pmc` for an `array` generated from 100 Monte-Carlo runs. `freq` specifies the operating frequency at which the pattern is computed.

`pmc = perturbedPattern(array,freq,az)` also specifies the azimuth angles `az` used for computing the perturbed patterns.

`pmc = perturbedPattern(array,freq,az,el)` also specifies the elevation angles `el` used for computing the perturbed patterns.

`pmc = perturbedPattern( ___ ,Name=Value)` also sets the specified parameter Name to the specified Value. You can specify additional name-value pair arguments in any order as (Name1=Value1,...,NameN=ValueN). For example, `NumTrials = 10000`.

`[pmc,pnm,mpmc,varpmc] = perturbedPattern(array, ___ )` also returns the nominal array response pattern `pnm`, the mean array response pattern `mpmc`, and the variance of the array response pattern `varpmc`.

`perturbedPattern(array, ___ )` plots all perturbed patterns for all Monte-Carlo runs and overlays them with both the nominal array response pattern and the mean perturbed array response pattern.

Examples

Compute Power Pattern of Perturbed ULA for All Azimuth Angles

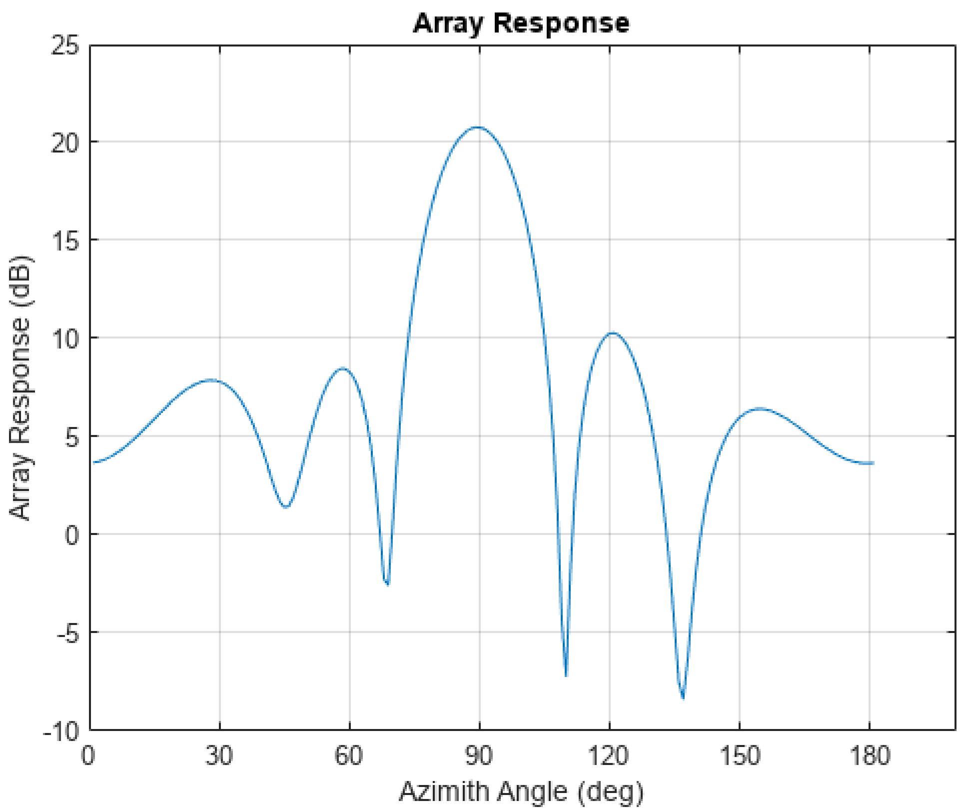
Create an 11-element Uniform Linear Array. The array operates at 300 MHz and its elements are spaced one-half wavelength apart. Perturb the element positions by 1/20th of a wavelength. Use the default value of 100 Monte Carlo runs. Compute the pattern for the first simulation at all azimuth angles. Set the random number generator seed for reproducibility.

```
freq = 300.0e6;
lambda = physconst('LightSpeed')/freq;
d = lambda/2;
array = phased.ULA(11,ElementSpacing=d/2);
```

```
rng(10007)
perturbations(array,'ElementPosition','Normal',0,lambda/20);
perturbations(array,'TaperMagnitude','Normal',0,0.1);
pmc = perturbedPattern(array,freq);
```

Plot the array pattern vs azimuth angle.

```
plot(mag2db(abs(pmc(:,1))))
ylabel('Array Response (dB)')
xlabel('Azimith Angle (deg)')
title('Array Response')
grid on
xticks([0:30:180])
```



Compute Power Pattern of Perturbed ULA for Span of Azimuth Angles

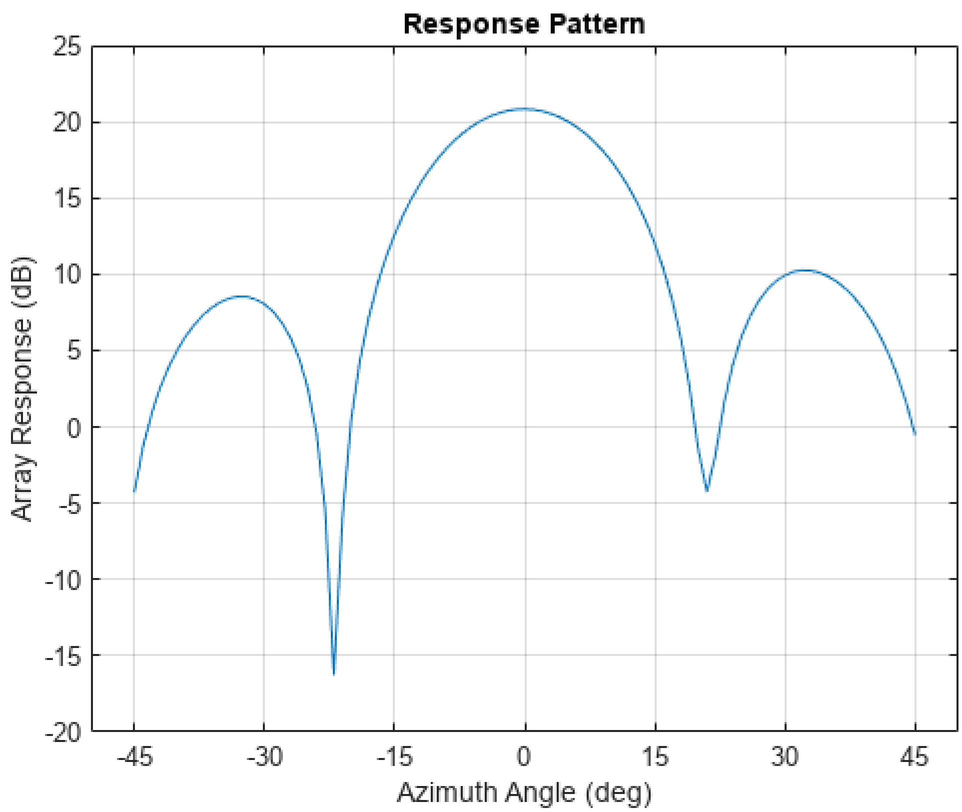
Create an 11-element uniform linear array (ULA). The array operates at 300 MHz and its elements are spaced one-half wavelength apart. Perturb the element positions by 1/20th of a wavelength. Use the default value of 100 Monte Carlo runs. Compute the pattern for the first simulation at azimuth angles from −45 to +45 degrees. Set the seed of the random number generator for reproducibilty.

```
freq = 300.0e6;
lambda = physconst('LightSpeed')/freq;
d = lambda/2;
array = phased.ULA(11,ElementSpacing = d/2);
```

```
rng(230081)
perturbations(array,'ElementPosition','Normal',0,lambda/20);
perturbations(array,'TaperMagnitude','Normal',0,0.1);
```

Display the array response pattern for the first Monte Carlo run.

```
azang = -45:45;
pmc = perturbedPattern(array,freq,azang);
plot(azang,mag2db(abs(pmc(:,1))))
title('Response Pattern')
xlabel('Azimuth Angle (deg)')
ylabel('Array Response (dB)')
xticks([-45:15:45])
grid on
```

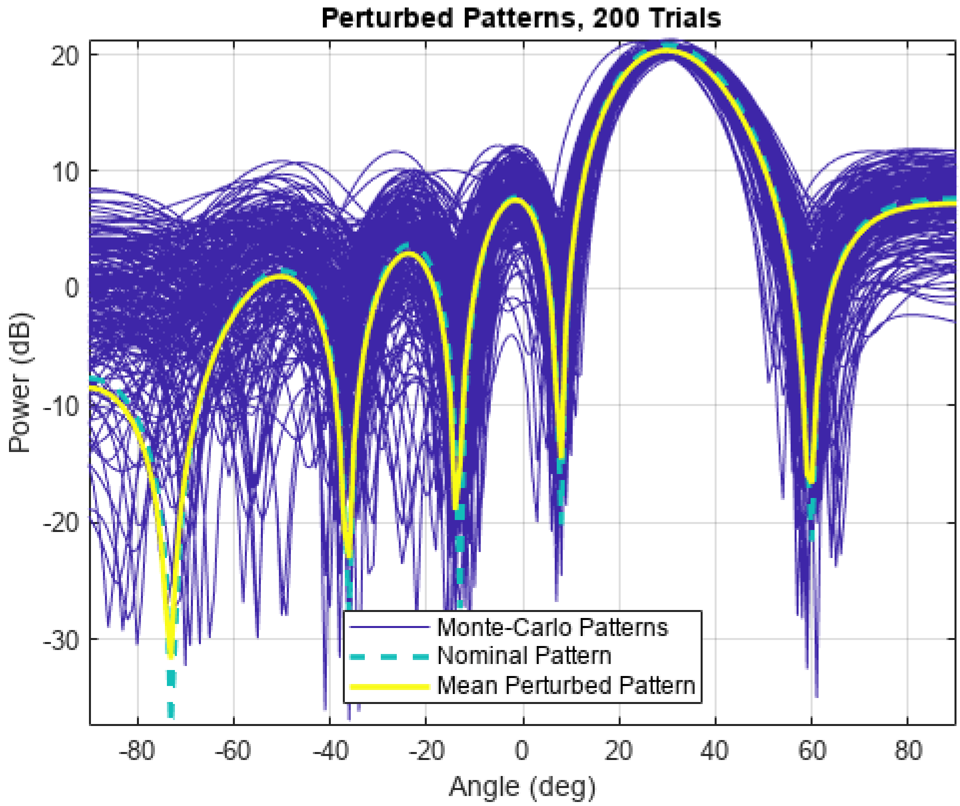


Compute Power Pattern for Steered Array

Create an 11-element Uniform Linear Array. The array operates at 300 MHz and its elements are spaced one-half wavelength apart. Perturb the element positions by 1/20th of a wavelength. Run 200 Monte Carlo simulations and steer the array 30 degrees in azimuth.

```
freq = 300e6;
lambda = physconst('Lightspeed')/freq;
d = lambda/2;
array = phased.ULA(11,ElementSpacing=d/2);
```

```
pos = getElementPosition(array);
sv = steervec(pos,30);
perturbations(array,'ElementPosition','Normal',0,lambda/20);
perturbations(array,'TaperMagnitude','Normal',0,0.1);
azang = -90:90;
elang = 0;
perturbedPattern(array,freq,azang,elang,Weights=sv,NumTrials=200);
```



Input Arguments

array — Phased array
Phased Array System Toolbox System object

Phased array, specified as a Phased Array System Toolbox System object.

freq — Frequency used to compute array patterns
positive scalar

Frequency used to compute array patterns, specified as a positive scalar. Units are in Hz.

Data Types: double

az — Azimuth angles
scalar | length-*M* real-valued vector

Azimuth angles for computing power pattern, specified as a scalar or length-*M* real-valued vector. Either `az` or `el`, must be a scalar. Units are in degrees.

Data Types: `double`

`el` — Elevation angles
scalar | length-*M* real-valued vector

Elevation angles for computing power pattern, specified as a scalar or length-*N* real-valued vector. Either `az` or `el` must be a scalar. Units are in degrees.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `pmc = perturbedPatter(array,NumTrials = 1000)`

`NumTrials` — Number of Monte-Carlo trials
positive integer

Number of Monte-Carlo trials, specified as a positive integer.

Data Types: `double`

`Parent` — Axes on which the pattern is displayed
axes handle

Axes on which the pattern is displayed, specified as an axis handle..

Data Types: `double`

`PropagationSpeed` — Signal propagation speed
`physconst('lightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value is `physconst('lightSpeed')`. Units are in m/s.

Example: `300e6`

Data Types: `double`

`Weights` — Element weights
length-*P* complex-valued vector

Element weights, specified as a length-*P* complex-valued vector. *P* is the number of elements belonging to the array specified in the `array` argument.

Example: `ones(11,1)`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

pmc — Perturbed array pattern
length-*Q* vector

Perturbed array pattern, returned as an *M*-by-*Q* or *N*-by-*Q* complex-valued matrix. *M* is the number of angles in the `az` argument. *N* is the number of angles in the *el* argument. Either `az` or `el` must be a scalar.*Q* is the number of Monte-Carlo trials. Each trial generates a different pattern. Units are in degrees.

Data Types: `double`
Complex Number Support: Yes

pnm — Nominal array pattern
length-*M* real-valued vector | length-*N* real-valued vector

Nominal array pattern, returned as an *M* complex-valued vector or as an *N*-by-*Q* complex-valued vector. *M* is the number of angles in the `az` argument. *N* is the number of angles in the `al` argument. Either `az` or `el` must be a scalar. Units are in degrees.

Data Types: `double`

mpmc — Mean of array response pattern
length-*N* real-valued vector | length-*N* real-valued vector

Mean of array response pattern, returned as a length-*M* real-valued vector or as an length-*N* real-valued vector. *M* is the number of angles in the `az` argument. *N* is the number of angles in the `al` argument. Units are in degrees.

Data Types: `double`

varpmc — Variance of array response pattern
length-*N* real-valued vector | length-*N* real-valued vector

Variance of array response pattern, returned as a length-*M* real-valued vector or as an length-*N* real-valued vector. *M* is the number of angles in the `az` argument. *N* is the number of angles in the `al` argument. Units are in degrees.

Data Types: `double`

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.ConformalArray` | `phased.HeterogeneousConformalArray` | `perturbedArray` | `perturbations`

plotGratingLobeDiagram

Plot grating lobe diagram of array

Syntax

```
plotGratingLobeDiagram(array,freq)
plotGratingLobeDiagram(array,freq,c)
plotGratingLobeDiagram(array,freq,angle,c)
plotGratingLobeDiagram(array,freq,angle,c,f0)
hndl = plotGratingLobeDiagram( ___ )
```

Description

`plotGratingLobeDiagram(array,freq)` plots the grating lobe diagram of an array in the *u-v* coordinate system. The System object `array` specifies the array. The argument `freq` specifies the signal frequency. The array, by default, is steered to 0° azimuth and 0° elevation.

A grating lobe diagram displays the positions of the peaks of the narrowband array pattern. The array pattern depends only upon the geometry of the array and not upon the types of elements which make up the array. Visible and non-visible grating lobes are displayed as open circles. Only grating lobe peaks near the location of the mainlobe are shown. The mainlobe itself is displayed as a filled circle.

`plotGratingLobeDiagram(array,freq,c)`, in addition, specifies the array steering angle, `angle`.

`plotGratingLobeDiagram(array,freq,angle,c)`, in addition, specifies the propagation speed by `c`.

`plotGratingLobeDiagram(array,freq,angle,c,f0)`, in addition, specifies an array phase-shifter frequency, `f0`, that differs from the signal frequency, `freq`. This argument is useful when the signal no longer satisfies the narrowband assumption and allows you to estimate the size of beam squint.

`hndl = plotGratingLobeDiagram( ___ )` returns the handle `hndl` to the plot for any of the input syntaxes.

Examples

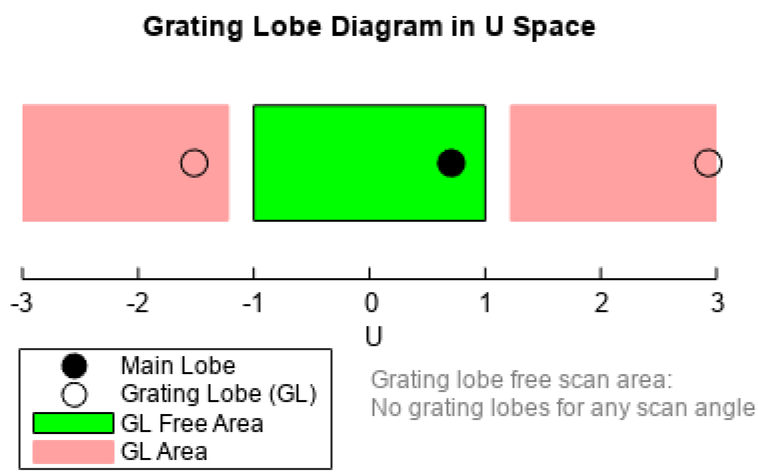
Create Grating Lobe Diagram for ULA

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing less than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 3 GHz and the spacing between elements is 0.45 of the wavelength. All elements are isotropic antenna elements. Steer the array in the direction 45 degrees in azimuth and 0 degrees in elevation.

```
c = physconst('LightSpeed');
f = 3e9;
```

```
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,...
    'ElementSpacing',0.45*lambda);
plotGratingLobeDiagram(sULA,f,[45;0],c);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region is defined by the direction cosine limits between [-1,1] and is marked by the two vertical black lines. Because the array spacing is less than one-half wavelength, there are no grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it coincides with the visible region.

The white area of the diagram indicates a region where no grating lobes are possible.

Input Arguments

`array` — Antenna, microphone, or transducer phased array
phased array

Antenna, microphone, or transducer phased array, specified as a System object.

freq — Frequency for computing directivity and patterns
positive scalar | 1-by-*L* real-valued row vector

Frequencies for computing directivity and patterns, specified as a positive scalar or 1-by-*L* real-valued row vector. Frequency units are in hertz.

- For an antenna, microphone, or sonar hydrophone or projector element, FREQ must lie within the range of values specified by the FrequencyRange or FrequencyVector property of the element. Otherwise, the element produces no response and the directivity is returned as –Inf. Most elements use the FrequencyRange property except for phased.CustomAntennaElement and phased.CustomMicrophoneElement, which use the FrequencyVector property.
- For an array of elements, FREQ must lie within the frequency range of the elements that make up the array. Otherwise, the array produces no response and the directivity is returned as –Inf.

Example: [1e8 2e6]

Data Types: double

angle — array steering angle
[0;0] (default) | 2-by-1 real-valued vector | scalar

Array steering angle, specified as either a 2-by-1 vector or a scalar. If angle is a vector, it takes the form [azimuth;elevation]. The azimuth angle must lie in the range [-180°,180°]. The elevation angle must lie in the range [-90°,90°]. All angle values are specified in degrees. If the argument angle is a scalar, it specifies only the azimuth angle where the corresponding elevation angle is 0°.

Data Types: double

c — signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are m/s.

Data Types: double

f0 — Phase-shifter frequency of array
freq (default) | scalar

Phase-shifter frequency of the array, specified as a scalar. When this argument is omitted, the phase-shifter frequency is assumed to be the signal frequency, freq. Units are Hz

Data Types: double

Output Arguments

hndl — plot handle
handle

Plot handle.

Algorithms

Grating Lobes

Spatial undersampling of a wavefield by an array gives rise to visible grating lobes. If you think of the wavenumber, k, as analogous to angular frequency, then you must sample the signal at spatial intervals smaller than π/k_{max} (or $\lambda_{min}/2$) in order to remove aliasing. The appearance of visible grating lobes is also known as spatial aliasing. The variable k_{max} is the largest wavenumber value present in the signal.

The directions of maximum spatial response of a ULA are determined by the peaks of the array's array pattern (alternatively called the beam pattern or array factor). Peaks other than the mainlobe peak are called grating lobes. For a ULA, the array pattern depends only on the wavenumber component of the wavefield along the array axis (the y-direction for the `phased.ULA` System object). The wavenumber component is related to the look-direction of an arriving wavefield by $k_y = -2\pi \sin \varphi/\lambda$. The angle φ is the broadside angle—the angle that the look-direction makes with a plane perpendicular to the array. The look-direction points away from the array to the wavefield source.

The array pattern possesses an infinite number of periodically-spaced peaks that are equal in strength to the mainlobe peak. If you steer the array to the φ_0 direction, the array pattern for a ULA has its mainlobe peak at the wavenumber value of $k_{y0} = -2\pi \sin \varphi_0/\lambda$. The array pattern has strong grating lobe peaks at $k_{ym} = k_{y0} + 2\pi\, m/d$, for any integer value m. Expressed in terms of direction cosines, the grating lobes occur at $u_m = u_0 + m\lambda/d$, where $u_0 = \sin \varphi_0$. The direction cosine, u_0, is the cosine of the angle that the look-direction makes with the y-axis and is equal to $\sin \varphi_0$ when expressed in terms of the look-direction.

In order to correspond to a physical look-direction, u_m must satisfy, $-1 \le u_m \le 1$. You can compute a physical look-direction angle φ_m from $\sin \varphi_m = u_m$ as long as $-1 \le u_m \le 1$. The spacing of grating lobes depends upon λ/d. When λ/d is small enough, multiple grating lobe peaks can correspond to physical look-directions.

The presence or absence of visible grating lobes for the ULA is summarized in this table.

Element Spacing	Grating Lobes
$\lambda/d \ge 2$	No visible grating lobes for any mainlobe direction.
$1 \le \lambda/d < 2$	Visible grating lobes can exist for some range of mainlobe directions.
$\lambda/d < 1$	Visible grating lobes exist for every mainlobe direction.

Version History

Introduced in R2022b

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`uv2azel` | `azel2uv`

viewArray

View array geometry

Syntax

```
viewArray(array)
viewArray(array,Name,Value)
hndl = viewArray( ___ )
```

Description

`viewArray(array)` displays the geometry of the `array`.

`viewArray(array,Name,Value)` displays the geometry of the array, with additional options specified by one or more `Name,Value` pair arguments.

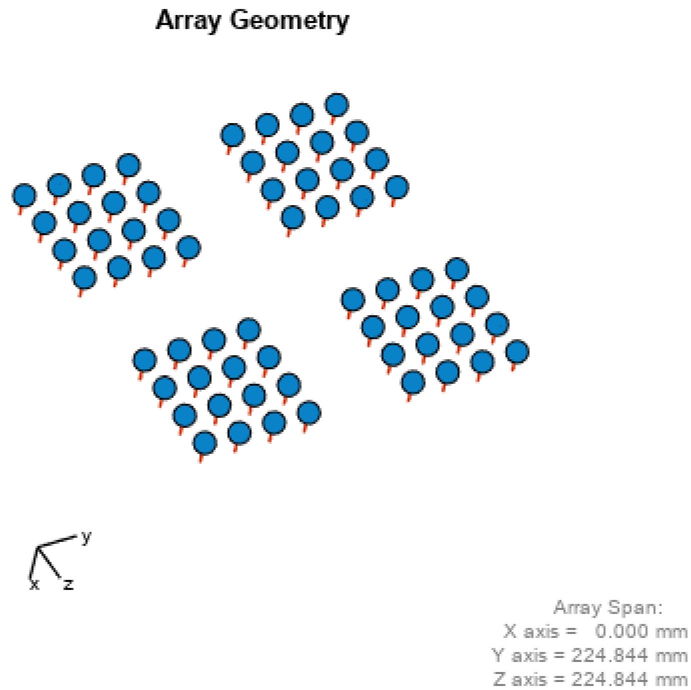
`hndl = viewArray( ___ )` returns the handle of the array elements in the figure window. All input arguments described for the previous syntaxes also apply here.

Examples

Geometry of NR Rectangular Panel Array

Construct a 5G antenna array where the grid is 2-by-2 and each panel is 4-by-4 array. Each antenna element consists of two short-dipole antennas with different dipole axis directions. The antenna elements are spaced 1/2 wavelength apart and the panels are spaced 3 wavelengths apart. The array operates at 6 GHz.

```
c = physconst('LightSpeed');
fc = 6.0e9;
lambda = c/fc;
antenna1 = phased.ShortDipoleAntennaElement('AxisDirection','Z');
antenna2 = phased.ShortDipoleAntennaElement('AxisDirection','X');
array = phased.NRRectangularPanelArray('ElementSet', ...
    {antenna1, antenna2},'Size',[4, 4, 2, 2], ...
    'Spacing',[0.5*lambda,0.5*lambda,3*lambda,3*lambda]);
viewArray(array,'ShowNormals',true, ...
    'ShowLocalCoordinates',true,'Orientation',[60;100;45], ...
    'ShowAnnotation',true)
```



Input Arguments

`array` — Phased array
System object

Phased array, specified as a System object.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'ShowNormals',true,'ShowIndex','All','ShowTaper',true`

`Parent` — Handle to axes
real-valued scalar

Handle to the axes along which the array geometry is displayed.

`ShowNormals` — Option to show normal vectors
`false` (default) | `true`

Option to show normal directions, specified as the comma-separated pair consisting of `'ShowNormals'` and a Boolean value.

- `true` — show the normal directions of all elements in the array
- `false` — plot the elements without showing normal directions

Example: `false`

Data Types: `logical`

`ShowLocalCoordinates` — Show local coordinates
`true` (default) | `false`

Logical flag specifying whether to show the local coordinate axes.

Data Types: `logical`

`ShowAnnotation` — Show aperture size and element spacing annotations
`true` (default) | `false`

Logical flag specifying whether to show the annotations in the UI panel of the figure. Annotation shows aperture size and element spacing based on array axis of array.

Data Types: `logical`

`Orientation` — Orientation of the array
`[0;0;0]` (default) | 3-by-1 column vector with real values

Orientation of the array, specified as a 3-by-1 column vector containing the rotation angles with respect to the x-, y-, and z-axes of the local coordinate system, respectively. The default value is [0;0;0].

Data Types: `double`

`ShowTaper` — Option to show taper magnitude
`false` (default) | `true`

Option to show taper magnitude, specified as the comma-separated pair consisting of `'ShowTaper'` and a Boolean value.

- `true` — change the element color brightness in proportion to the element taper magnitude
- `false` — plot all elements using the same color

Example: `true`

Data Types: `logical`

`ShowIndex` — Element indices to show
`'None'` (default) | vector of positive integers | `'All'`

Element indices to show in the figure, specified as the comma-separated pair consisting of `'ShowIndex'` and a vector of positive integers. Each number in the vector must be an integer between 1 and the number of elements. To show all of indices of the array, specify `'All'`. To suppress all indices, specify `'None'`.

Example: `[1,2,3]`

Data Types: `double`

`Title` — Plot title
`'Array Geometry'` (default) | character vector

Plot title, specified as a character vector.

Example: `'My array plot'`

Data Types: `char` | `string`

Output Arguments

`hndl` — Handle of array elements plot
scalar

Handle of array elements plot in the figure window, returned as a scalar.

Version History

Introduced in R2021a

bandwidth

Package: `phased`

Waveform bandwidth

Syntax

`BW = bandwidth(waveform)`

Description

`BW = bandwidth(waveform)` returns the bandwidth `BW` of the pulses for the `waveform` System object.

- For a linear FM waveform, the bandwidth equals the value of the `SweepBandwidth` property.
- For a rectangular waveform, the bandwidth equals the reciprocal of the pulse width.
- For the stepped FM waveform, If there are N frequency steps, the bandwidth equals N times the value of the `FrequencyStep` property. If there is no frequency stepping, the bandwidth equals the reciprocal of the pulse width.
- For phased coded waveform, The bandwidth value is the reciprocal of the chip width.

Examples

Compute Linear FM Bandwidth

Determine the bandwidth of a linear FM pulse waveform. The default value for an LFM waveform is 100 kHz.

```
waveform = phased.LinearFMWaveform;
bw = bandwidth(waveform)
```

```
bw = 100000
```

Input Arguments

`waveform` — Waveform object
waveform System object

Waveform, specified as a waveform System object.

Output Arguments

`BW` — Pulse bandwidth
scalar

Pulse bandwidth, returned as a positive scalar. Units are in Hz.

Version History

Introduced in R2023a

getStretchProcessor

Package: `phased`

Create stretch processor for waveform

Syntax

```
strproc = getStretchProcessor(waveform)
strproc = getStretchProcessor(waveform,refrng)
strproc = getStretchProcessor(waveform,refrng,rngspan)
strproc = getStretchProcessor(waveform,refrng,rngspan,v)
```

Description

`strproc = getStretchProcessor(waveform)` returns the stretch processor `strproc` for the `waveform` System object. By default the processor is set up so the reference range corresponds to 1/4 of the maximum unambiguous range of a pulse. The range span corresponds to 1/10 of the distance traveled by the wave within the pulse width. The propagation speed is the speed of light.

`strproc = getStretchProcessor(waveform,refrng)` also specifies the reference range.

`strproc = getStretchProcessor(waveform,refrng,rngspan)` also specifies the range span `rngspan`. The reference interval is centered at `refrng`.

`strproc = getStretchProcessor(waveform,refrng,rngspan,v)` specifies the propagation speed `v`.

Examples

Detect a Target Using Stretch Processing

Use stretch processing to locate a target at a range of 4950 m.

Simulate the signal.

```
waveform = phased.LinearFMWaveform;
x = waveform();
c = physconst('LightSpeed');
rng = 4950.0;
num_samples = round(rng/(c/(2*waveform.SampleRate)));
x = circshift(x,num_samples);
```

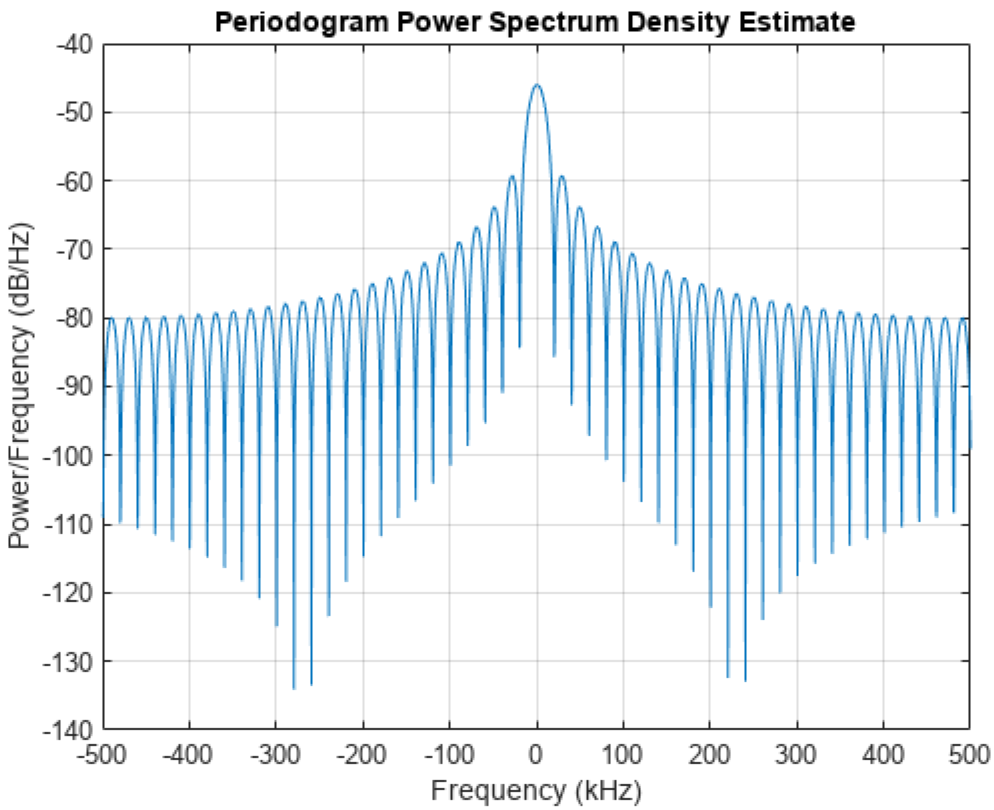
Perform stretch processing.

```
stretchproc = getStretchProcessor(waveform,5000,200,c);
y = stretchproc(x);
```

Plot the spectrum of the resulting signal.

```
[Pxx,F] = periodogram(y,[],2048,stretchproc.SampleRate,'centered');
plot(F/1000,10*log10(Pxx))
```

```
grid
xlabel('Frequency (kHz)')
ylabel('Power/Frequency (dB/Hz)')
title('Periodogram Power Spectrum Density Estimate')
```



Detect the range.

```
[~,rngidx] = findpeaks(pow2db(Pxx/max(Pxx)),'MinPeakHeight',-5);
rngfreq = F(rngidx);
rng = stretchfreq2rng(rngfreq,stretchproc.SweepSlope,stretchproc.ReferenceRange,c)
```

```
rng = 4.9634e+03
```

Input Arguments

`waveform` — Waveform
pulse waveform System object

Pulse waveform, specified as a Phased Array System Toolbox System object.

Example: `phased.RectangularWaveform`

`refrng` — Reference range
1/4 of the maximum unambiguous range of a pulse (default) | positive scalar

Reference range, specified as a positive scalar. Units are in meters.

Data Types: `double`

`rngspan` — Range span of interest
1/10 of the distance traveled by the wave within the pulse width (default) | positive scalar

Range span of interest, specfied as a positive scalar. The center of the interval is the range value specified in the `refrng` argument. Units are in meters.

Data Types: `double`

`v` — Propagation speed
speed of light (default) | positive scalar

Propagation speed, specified as a positive scalar. Units are in meters per second.

Example: `300.0`

Data Types: `double`

Output Arguments

`strproc` — Stretch processor
stretch processor System object

Stretch processor, returned as a `phased.StretchProcessor` System object.

Version History

Introduced in R2023a

See Also

`phased.StretchProcessor` | `stretchfreq2rng`

Topics

Range Estimation Using Stretch Processing
"Stretch Processing"

getMatchedFilter

Package: phased

Matched filter coefficients derived from waveform

Syntax

```
coeff = getMatchedFilter(waveform)
coeff = getMatchedFilter(waveform,FrequencyOffset = foffset)
```

Description

`coeff = getMatchedFilter(waveform)` returns the matched filter coefficients `coeff` for the pulse FM `waveform` object.

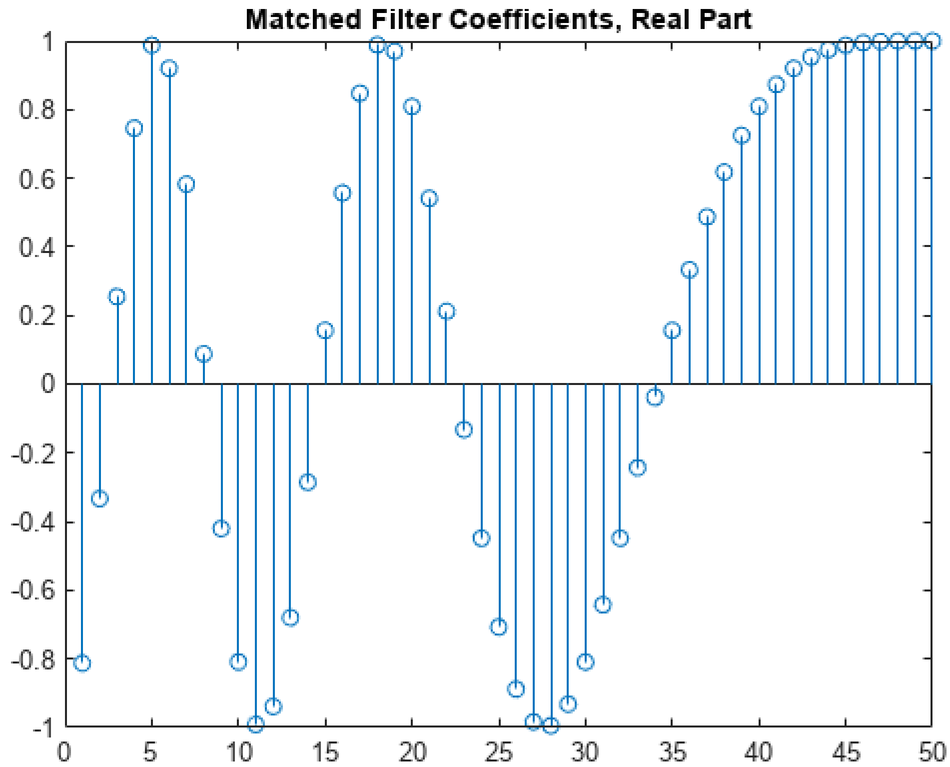
`coeff = getMatchedFilter(waveform,FrequencyOffset = foffset)` also specifies a frequency offset `foffset` when matched filter coefficients are generated. To enable this syntax, set the `FrequencyOffsetSource` property of the `waveform` to `'Input port'`.

Examples

Matched Filter Coefficients of Linear FM Waveform

Get the matched filter coefficients for a linear FM pulse.

```
waveform = phased.LinearFMWaveform('PulseWidth',5e-05,...
    'SweepBandwidth',1e5,'OutputFormat','Pulses');
coeff = getMatchedFilter(waveform);
stem(real(coeff))
title('Matched Filter Coefficients, Real Part')
```



Input Arguments

`waveform` — Pulse waveform

pulse waveform System object

Pulse waveform, specified as a waveform System object.

Example: `phased.RectangularWaveform`

`foffset` — frequency offset

scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `12000.0`

Data Types: `double`

Output Arguments

`coeff` — Matched filter coefficients

complex-valued column vector

Matched filter coefficients, returned as a complex-valued column vector. Units are dimensionless.

Version History

Introduced in R2023a

plot

Package: `phased`

Plot pulse waveform

Syntax

```
plot(waveform)
plot(waveform,Name=Value)
plot(waveform,Name=Value,LineSpec)
hndl = plot( ___ )
```

Description

`plot(waveform)` plots the real part of the `waveform`.

`plot(waveform,Name=Value)` plots the waveform with additional options specified by one or more (`Name=Value`) pair arguments.

`plot(waveform,Name=Value,LineSpec)` specifies the same line color, line style, or marker options as are available in the MATLAB `plot` function.

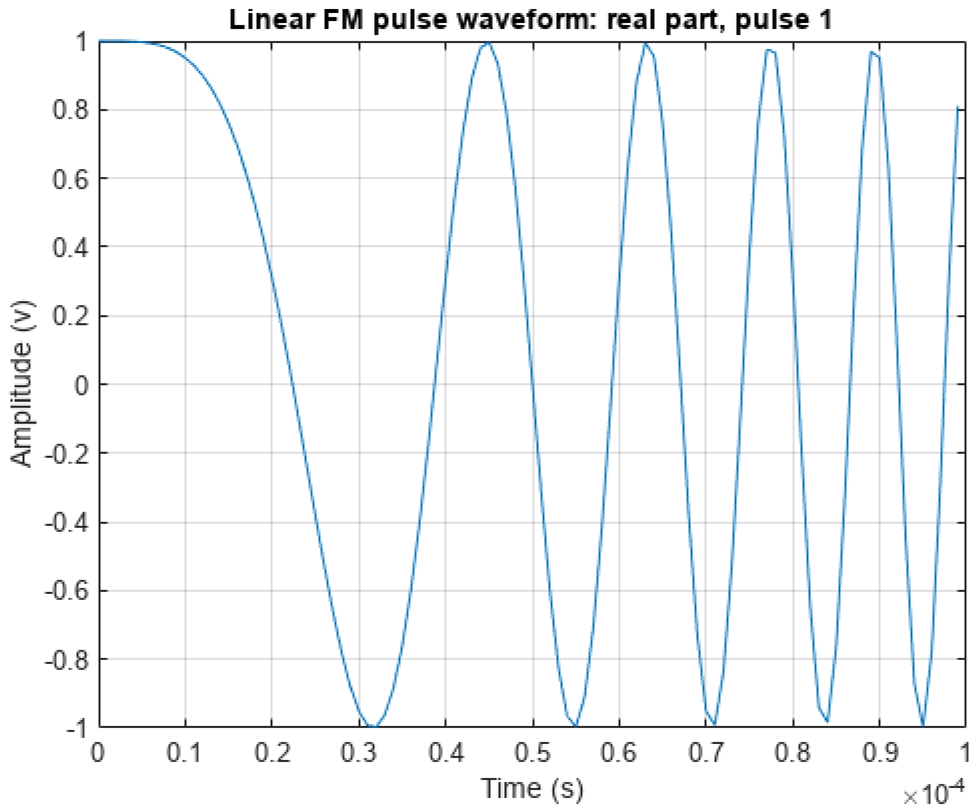
`hndl = plot( ___ )` returns the line handle in the figure.

Examples

Plot Linear FM Pulse

Create and plot an upsweep linear FM pulse waveform.

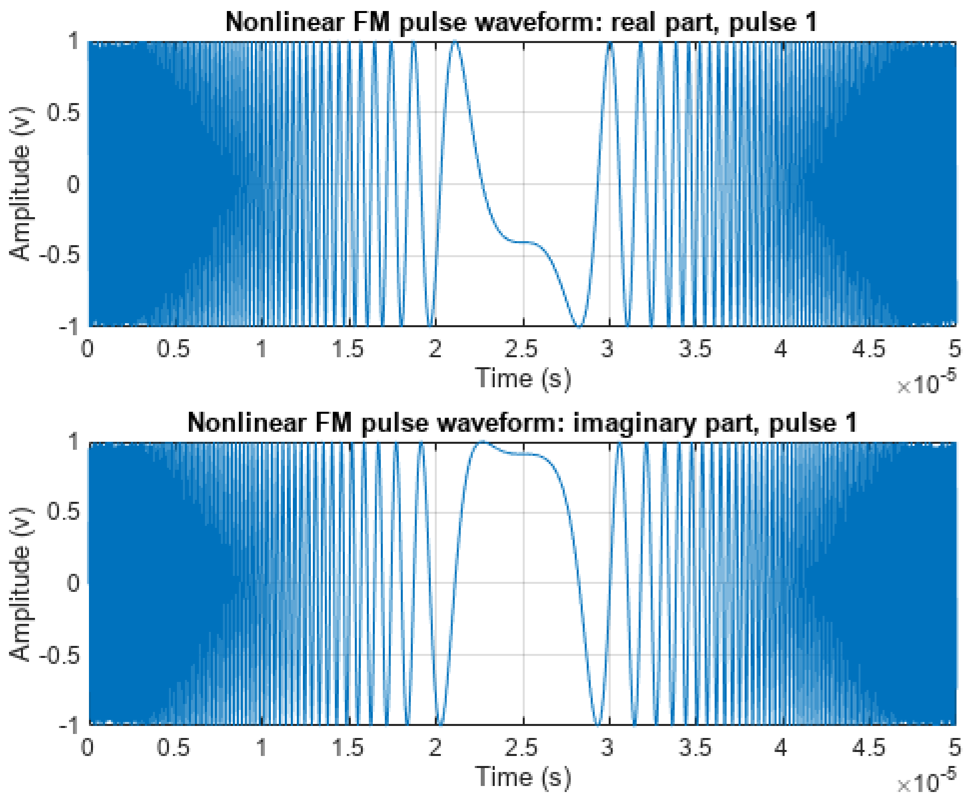
```
waveform = phased.LinearFMWaveform('SweepBandwidth',1e5,'PulseWidth',1e-4);
plot(waveform);
```



Plot Quadratic FM Pulse Waveform

Create and plot a quadratic FM pulse waveform. The pulse has a 10 MHz bandwidth and 50 μsec duration. The pulse sample rate is 10 times the bandwidth.

```
BW = 10e6;
T = 50e-6;
waveform = phased.NonlinearFMWaveform( ...
    'SampleRate',10*BW,'SweepBandwidth',BW, ...
    'PulseWidth',T);
plot(waveform,PlotType='complex')
```



Input Arguments

`waveform` — Pulse waveform
waveform System object

Pulse waveform, specified as a System object.

`LineSpec` — Line properties
`'b'` (default) | character vector

Character vector to specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `PlotType` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `plot(waveform,PlotType='complex',pulseidx=4)`

`PlotType` — plot type
`'real'` (default) | `'imag'` | `'complex'`

Specifies whether the function plots the real part, imaginary part, or both parts of the waveform. Valid values are `'real'`, `'imag'`, and `'complex'`.

Example: `'complex'`

Data Types: `char` | `string`

`pulseidx` — Index of pulse to plot
`1` (default) | positive integer

Index of the pulse to plot. This value must be a scalar.

Example: 4

Data Types: `double`

`FrequencyOffset` — Frequency offset
0.0 (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `-100`

Data Types: `double`

`FrequencyOffsetSource` — Frequency offset source
`'Input port'` (default)

Frequency offset source, specified as a scalar. Units are in Hz.

Example: `'Input port'`

Data Types: `double`

Output Arguments

`hndl` — Line handle
scalar | positive valued column vector

Handle to the line or lines in the figure. For a `PlotType` value of `'complex'`, `hndl` is a column vector. The first and second elements of this vector are the handles to the lines in the real and imaginary subplots, respectively.

Version History

Introduced in R2023a

Functions

aictest

Dimension of signal subspace

Syntax

```
nsig = aictest(X)
nsig = aictest(X,'fb')
```

Description

`nsig = aictest(X)` estimates the number of signals, `nsig`, present in a snapshot of data, `X`, that impinges upon the sensors in an array. The estimator uses the Akaike Information Criterion test (AIC). The input argument, `X`, is a complex-valued matrix containing a time sequence of data samples for each sensor. Each row corresponds to a single time sample for all sensors.

`nsig = aictest(X,'fb')` estimates the number of signals. Before estimating, it performs forward-backward averaging on the sample covariance matrix constructed from the data snapshot, `X`. This syntax can use any of the input arguments in the previous syntax.

Examples

Estimate the Signal Subspace Dimensions for Two Arriving Signals

Construct a data snapshot for two plane waves arriving at a half-wavelength-spaced uniform line array with 10 elements. The plane waves arrive from 0° and –25° azimuth, both with elevation angles of 0°. Assume the signals arrive in the presence of additive noise that is both temporally and spatially Gaussian white. For each signal, the SNR is 5 dB. Take 300 samples to build a 300-by-10 data snapshot. Then, solve for the number of signals using `aictest`.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25];
x = sensorsig(elementPos,300,angles,db2pow(-5));
nsig = aictest(x)
```

```
nsig = 2
```

The result shows that the number of signals is two, as expected.

Estimate the Signal Subspace Dimension Using Forward-Backward Smoothing

Construct a data snapshot for two plane waves arriving at a half-wavelength-spaced uniform line array with 10 elements. Two correlated plane waves arrive from 0° and 10° azimuth, both with elevation angles of 0°. Assume that the signals arrive in the presence of additive noise that is both temporally and spatially Gaussian white. For each signal, the SNR is 10 dB. Take 300 samples to build a 300-by-10 data snapshot. Then, solve for the number of signals using `aictest`.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 10];
ncov = db2pow(-10);
scov = [1 .5]'*[1 .5];
x = sensorsig(elementPos,300,angles,ncov,scov);
Nsig = aictest(x)
```

```
Nsig = 1
```

This result shows that `aictest` cannot determine the number of signals correctly when the signals are correlated.

Use the forward-backward smoothing option.

```
Nsig = aictest(x,'fb')
```

```
Nsig = 2
```

The addition of forward-backward smoothing yields the correct number of signals.

Input Arguments

X — Data snapshot
complex-valued *K*-by-*N* matrix

Data snapshot, specified as a complex-valued, *K*-by-*N* matrix. A snapshot is a sequence of time-samples taken simultaneous at each sensor. In this matrix, *K* represents the number of time samples of the data, while *N* represents the number of sensor elements.

Example: [–0.1211 + 1.2549i, 0.1415 + 1.6114i, 0.8932 + 0.9765i;]

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`nsig` — Dimension of signal subspace
non-negative integer

Dimension of signal subspace, returned as a non-negative integer. The dimension of the signal subspace is the number of signals in the data.

More About

Estimating the Number of Sources

AIC and MDL tests

Direction finding algorithms such as MUSIC and ESPRIT require knowledge of the number of sources of signals impinging on the array or equivalently, the dimension, *d*, of the signal subspace. The Akaike

Information Criterion (AIC) and the Minimum Description Length (MDL) formulas are two frequently-used estimators for obtaining that dimension. Both estimators assume that, besides the signals, the data contains spatially and temporally white Gaussian random noise. Finding the number of sources is equivalent to finding the multiplicity of the smallest eigenvalues of the sampled spatial covariance matrix. The sample spatial covariance matrix constructed from a data snapshot is used in place of the actual covariance matrix.

A requirement for both estimators is that the dimension of the signal subspace be less than the number of sensors, N, and that the number of time samples in the snapshot, K, be much greater than N.

A variant of each estimator exists when forward-backward averaging is employed to construct the spatial covariance matrix. Forward-backward averaging is useful for the case when some of the sources are highly correlated with each other. In that case, the spatial covariance matrix may be ill conditioned. Forward-backward averaging can only be used for certain types of symmetric arrays, called centro-symmetric arrays. Then the forward-backward covariance matrix can be constructed from the sample spatial covariance matrix, S, using $S_{FB} = S + JS^*J$ where J is the exchange matrix. The exchange matrix maps array elements into their symmetric counterparts. For a line array, it would be the identity matrix flipped from left to right.

All the estimators are based on a cost function

$$L_d(d) = K(N-d)\ln\left\{\frac{\frac{1}{N-d}\sum_{i=d+1}^{N}\widehat{\lambda}_i}{\left\{\prod_{i=d+1}^{N}\widehat{\lambda}_i\right\}^{\frac{1}{N-d}}}\right\}$$

plus an added penalty term. The value λ_i represent the smallest *(N–d)* eigenvalues of the spatial covariance matrix. For each specific estimator, the solution for d is given by

- AIC

  $$\widehat{d}_{AIC} = \underset{d}{\operatorname{argmin}}\ \{L_d(d) + d(2N-d)\}$$

- AIC for forward-backward averaged covariance matrices

  $$\widehat{d}_{AIC:FB} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + \frac{1}{2}d(2N-d+1)\right\}$$

- MDL

  $$\widehat{d}_{MDL} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + \frac{1}{2}(d(2N-d)+1)\ln K\right\}$$

- MDL for forward-backward averaged covariance matrices

  $$\widehat{d}_{MDL\,FB} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + \frac{1}{4}d(2N-d+1)\ln K\right\}$$

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`espritdoa` | `mdltest` | `rootmusicdoa` | `spsmooth`

albersheim

Required SNR using Albersheim's equation

Syntax

```
SNR = albersheim(Pd,Pfa)
SNR = albersheim(Pd,Pfa,N)
```

Description

`SNR = albersheim(Pd,Pfa)` returns the signal-to-noise ratio in decibels. This value indicates the ratio required to achieve the given probabilities of detection `Pd` and false alarm `Pfa` for a single sample.

`SNR = albersheim(Pd,Pfa,N)` determines the required SNR for the noncoherent integration of `N` samples.

Examples

Compute Required SNR for Probability of Detection

Compute the required SNR of a single pulse to achieve a detection probability of 0.9 as a function of the fals- alarm probability.

Set the probability of detection to 0.9 and the probabilities of false alarm from 0.0001 to 0.01.

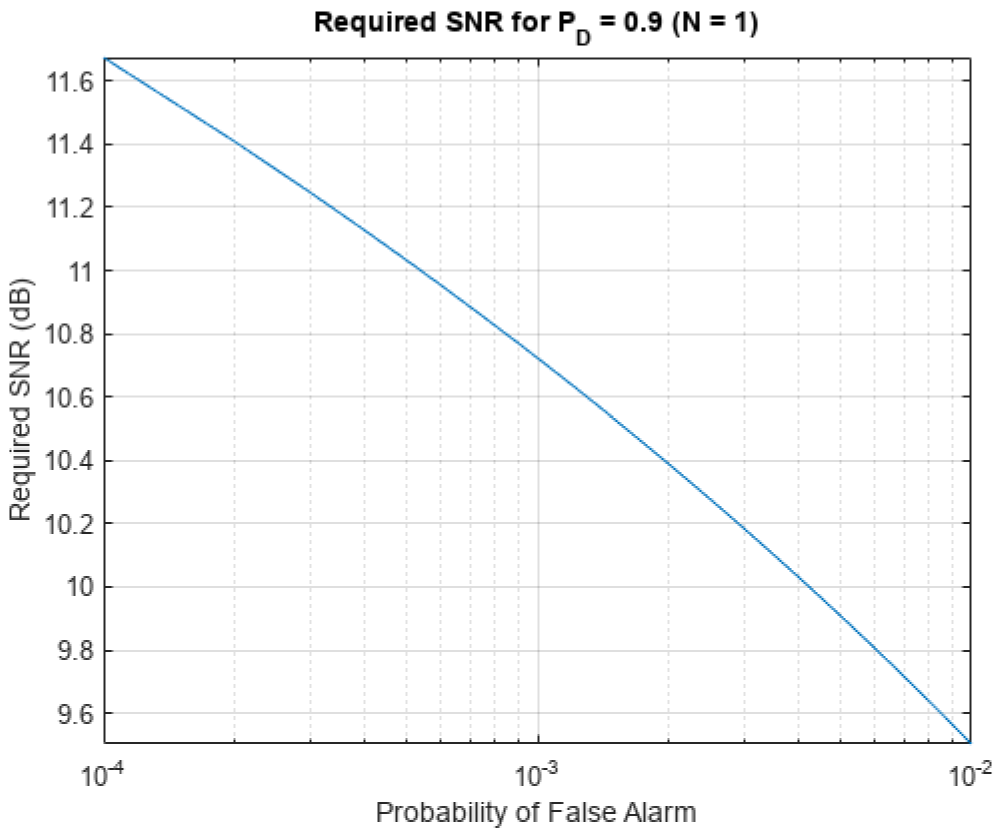
```
Pd = 0.9;
Pfa = 0.0001:0.0001:.01;
```

Loop the Albersheim equation over all false-alarm probabilities.

```
snr = zeros(1,length(Pfa));
for j = 1:length(Pfa)
    snr(j) = albersheim(Pd,Pfa(j));
end
```

Plot the SNR as a function of false-alarm probability.

```
semilogx(Pfa,snr)

grid
axis tight
xlabel("Probability of False Alarm")
ylabel("Required SNR (dB)")
title("Required SNR for P_D = "+Pd+" (N = 1)")
```



Compute Required SNR for Probability of Detection of 10 Pulses

Compute the required SNR of 10 noncoherently integrated pulse to achieve a detection probability of 0.9 as a function of the false-alarm probability.

Set the probability of detection to 0.9 and the probabilities of false alarm from 0.0001 to 0.01.

```
Pd = 0.9;
Pfa = 0.0001:0.0001:.01;
Npulses = 10;
```

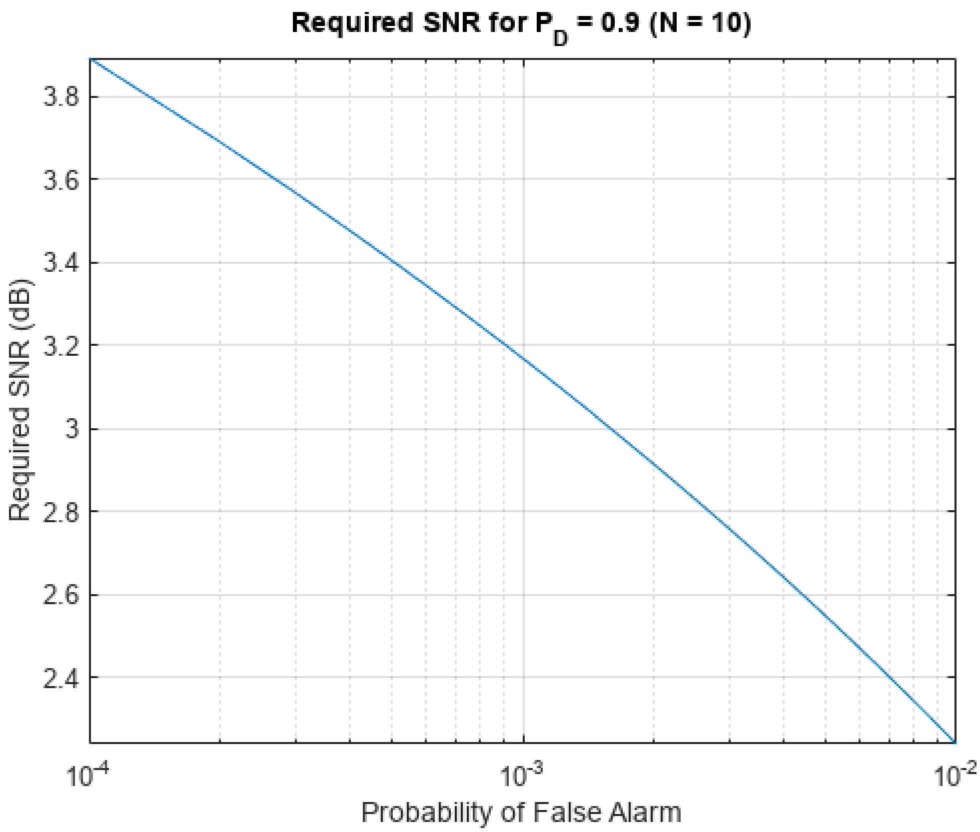
Loop over the Albersheim equation over all the false-alarm probabilities.

```
snr = zeros(1,length(Pfa));
for j = 1:length(Pfa)
    snr(j) = albersheim(Pd,Pfa(j),Npulses);
end
```

Plot the SNR as a function of the false-alarm probability.

```
semilogx(Pfa,snr)

grid
axis tight
xlabel("Probability of False Alarm")
```

```
ylabel("Required SNR (dB)")
title("Required SNR for P_D = "+Pd+" (N = 10)")
```



Input Arguments

Pd — Probability of detection
positive scalar

Probability of detection, specified as a positive scalar.

Data Types: `single` | `double`

`Pfa` — Probability of false alarm
positive scalar

Probability of false alarm, specified as a positive scalar.

Data Types: `single` | `double`

N — Number of pulses for noncoherent integration
1 (default) | positive scalar

Number of pulses for noncoherent integration, specified as a positive scalar.

Data Types: `single` | `double`

More About

Albersheim's Equation

Albersheim's equation uses a closed-form approximation to calculate the SNR. This SNR value is required to achieve the specified detection and false-alarm probabilities for a nonfluctuating target in independent and identically distributed Gaussian noise. The approximation is valid for a linear detector and is extensible to the noncoherent integration of N samples.

Let

$$A = \ln\frac{0.62}{P_{FA}}$$

and

$$B = \ln\frac{P_D}{1 - P_D}$$

where P_{FA} and P_D are the false-alarm and detection probabilities, respectively

Albersheim's equation for the required SNR in decibels is:

$$\mathrm{SNR} = -5\log_{10}N + [6.2 + 4.54/\sqrt{N + 0.44}]\log_{10}(A + 0.12AB + 1.7B)$$

where N is the number of noncoherently integrated samples.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

See Also

`shnidman`

ambgfun

Ambiguity and crossambiguity function

Syntax

```
afmag = ambgfun(x,Fs,PRF)
afmag = ambgfun(x,y,Fs,PRF)
[afmag,delay,doppler] = ambgfun( ___ )
[afmag,delay,doppler] = ambgfun( ___ ,"Cut","2D")
[afmag,delay] = ambgfun( ___ ,"Cut","Doppler")
[afmag,delay] = ambgfun( ___ ,"Cut","Doppler","CutValue",V)
[afmag,doppler] = ambgfun( ___ ,"Cut","Delay")
[afmag,doppler] = ambgfun( ___ ,"Cut","Delay","CutValue",V)
ambgfun( ___ )
```

Description

`afmag = ambgfun(x,Fs,PRF)` returns the magnitude of the normalized ambiguity function for the vector `x`. `Fs` is the sampling rate. `PRF` is the pulse repetition rate.

`afmag = ambgfun(x,y,Fs,PRF)` returns the magnitude of the normalized crossambiguity function between the pulse `x` and the pulse `y`.

`[afmag,delay,doppler] = ambgfun( ___ )` or `[afmag,delay,doppler] = ambgfun( ___ ,"Cut","2D")` returns the time delay vector, `delay`, and the Doppler frequency vector, `doppler`.

`[afmag,delay] = ambgfun( ___ ,"Cut","Doppler")` returns delays from a zero-Doppler cut through the 2-D normalized ambiguity function magnitude.

`[afmag,delay] = ambgfun( ___ ,"Cut","Doppler","CutValue",V)` returns delays from a nonzero Doppler cut through the 2-D normalized ambiguity function magnitude at Doppler value, `V`.

`[afmag,doppler] = ambgfun( ___ ,"Cut","Delay")` returns the Doppler values from zero-delay cut through the 2-D normalized ambiguity function magnitude.

`[afmag,doppler] = ambgfun( ___ ,"Cut","Delay","CutValue",V)` returns the Doppler values from a one-dimensional cut through the 2-D normalized ambiguity function magnitude at a delay value of `V`.

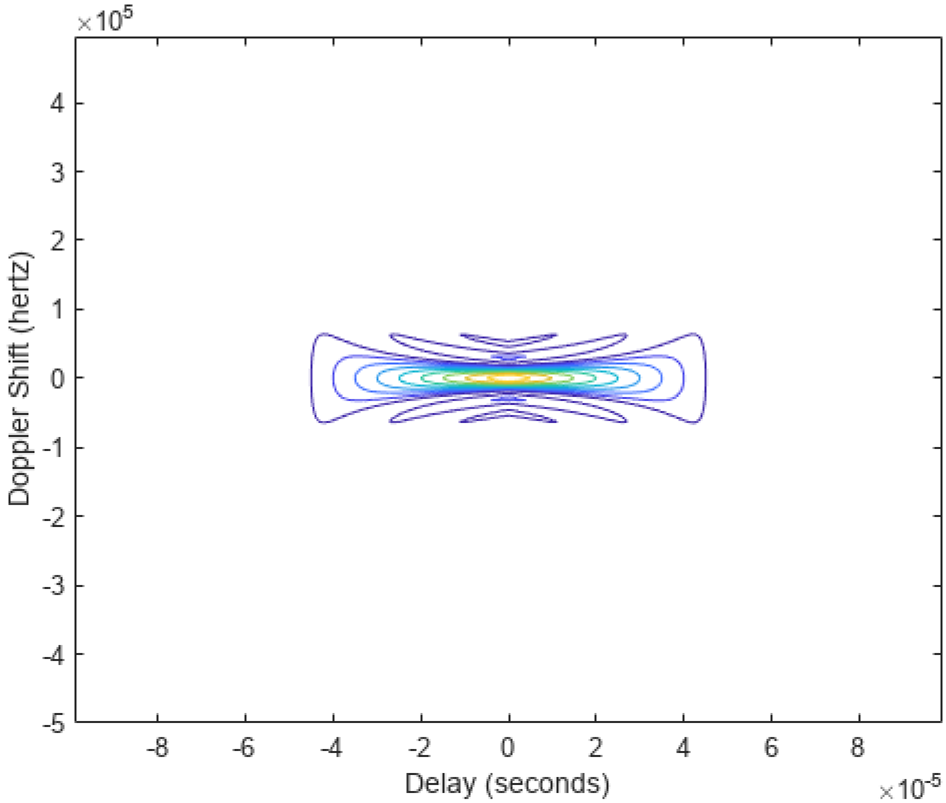
`ambgfun( ___ )`, with no output arguments, plots the ambiguity or crossambiguity function. When `"Cut"` is `"2D"`, the function produces a contour plot of the periodic ambiguity function. When `"Cut"` is `"Delay"` or `"Doppler"`, the function produces a line plot of the periodic ambiguity function cut.

Examples

Plot Ambiguity Function of Rectangular Pulse

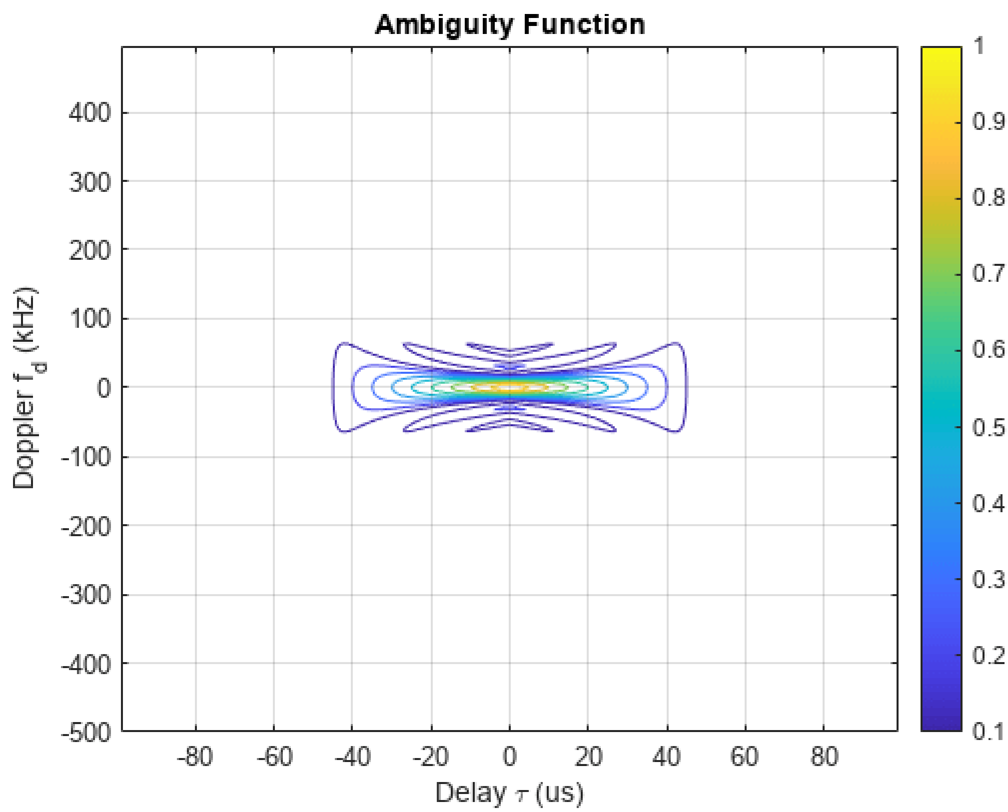
Plot the ambiguity function magnitude of a rectangular pulse.

```
waveform = phased.RectangularWaveform;
x = waveform();
PRF = waveform.PRF;
[afmag,delay,doppler] = ambgfun(x,waveform.SampleRate,PRF);
contour(delay,doppler,afmag)
xlabel("Delay (seconds)")
ylabel("Doppler Shift (hertz)")
```



Use the `ambgfun` function with no output arguments to recreate the plot.

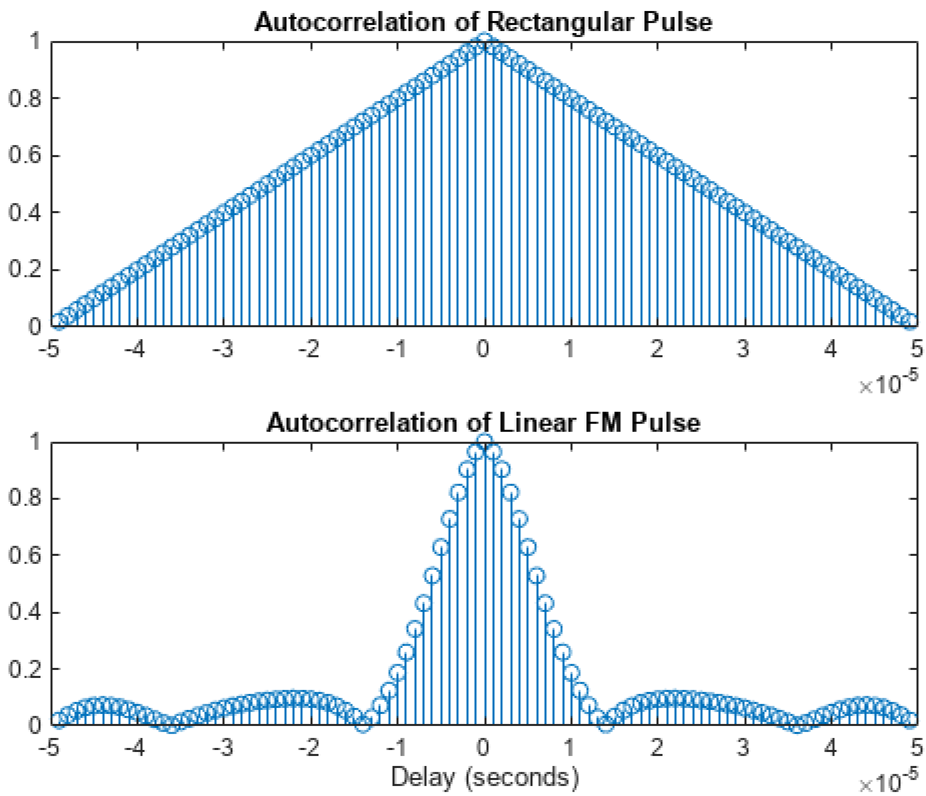
```
ambgfun(x,waveform.SampleRate,PRF)
```



Plot Autocorrelation Sequences of Rectangular and Linear FM Pulses

This example shows how to plot zero-Doppler cuts of the autocorrelation sequences of rectangular and linear FM pulses of equal duration. Note the pulse compression exhibited in the autocorrelation sequence of the linear FM pulse.

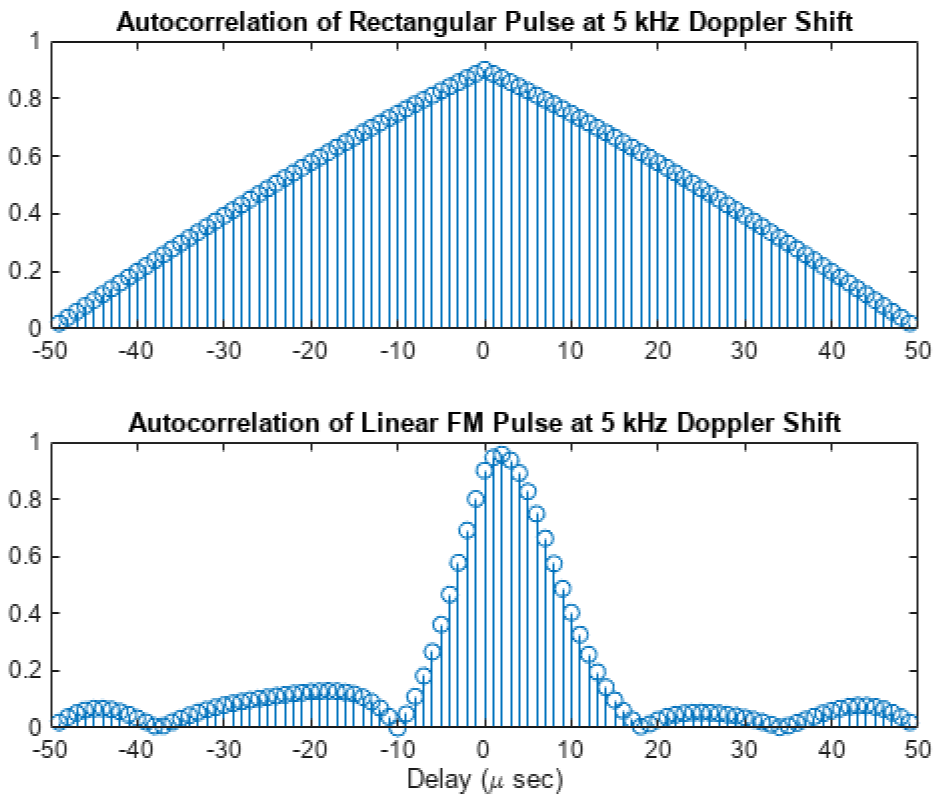
```
hrect = phased.RectangularWaveform("PRF",2e4);
hfm = phased.LinearFMWaveform("PRF",2e4);
xrect = hrect();
xfm = hfm();
[ambrect,delayrect] = ambgfun(xrect,hrect.SampleRate,...,
    hrect.PRF,"Cut","Doppler");
[ambfm,delayfm] = ambgfun(xfm,hfm.SampleRate,...,
    hfm.PRF,"Cut","Doppler");

subplot(2,1,1)
stem(delayrect,ambrect)
title("Autocorrelation of Rectangular Pulse")
subplot(2,1,2)
stem(delayfm,ambfm)
xlabel("Delay (seconds)")
title("Autocorrelation of Linear FM Pulse")
```



Plot Nonzero-Doppler Cuts of Autocorrelation Sequences

Plot nonzero-Doppler cuts of the autocorrelation sequences of rectangular and linear FM pulses of equal duration. Both cuts are taken at a 5 kHz Doppler shift. Besides the reduction of the peak value, there is a strong shift in the position of the linear FM peak, evidence of range-doppler coupling.

```
hrect = phased.RectangularWaveform("PRF",2e4);
hfm = phased.LinearFMWaveform("PRF",2e4);
xrect = hrect();
xfm = hfm();
fd = 5000;
[ambrect,delayrect] = ambgfun(xrect,hrect.SampleRate,...,
    hrect.PRF,"Cut","Doppler","CutValue",fd);
[ambfm,delayfm] = ambgfun(xfm,hfm.SampleRate,...,
    hfm.PRF,"Cut","Doppler","CutValue",fd);

figure
subplot(2,1,1)
stem(delayrect*10^6,ambrect)
title("Autocorrelation of Rectangular Pulse at 5 kHz Doppler Shift")
subplot(2,1,2)
stem(delayfm*10^6,ambfm)
xlabel("Delay (\mu sec)")
title("Autocorrelation of Linear FM Pulse at 5 kHz Doppler Shift")
```



Plot Crossambiguity Function

Plot the crossambiguity function between an LFM pulse and a delayed replica. Compare the crossambiguity function with the original ambiguity function. Set the sampling rate to 100 Hz, the pulse width to 0.5 sec, and the pulse repetition frequency to 1 Hz. The delay or lag is 10 samples equal to 100 ms. The bandwidth of the LFM signal is 10 Hz.

```
fs = 100.0;
bw1 = 10.0;
prf = 1;
nsamp = fs/prf;
pw = 0.5;
nlag = 10;
```

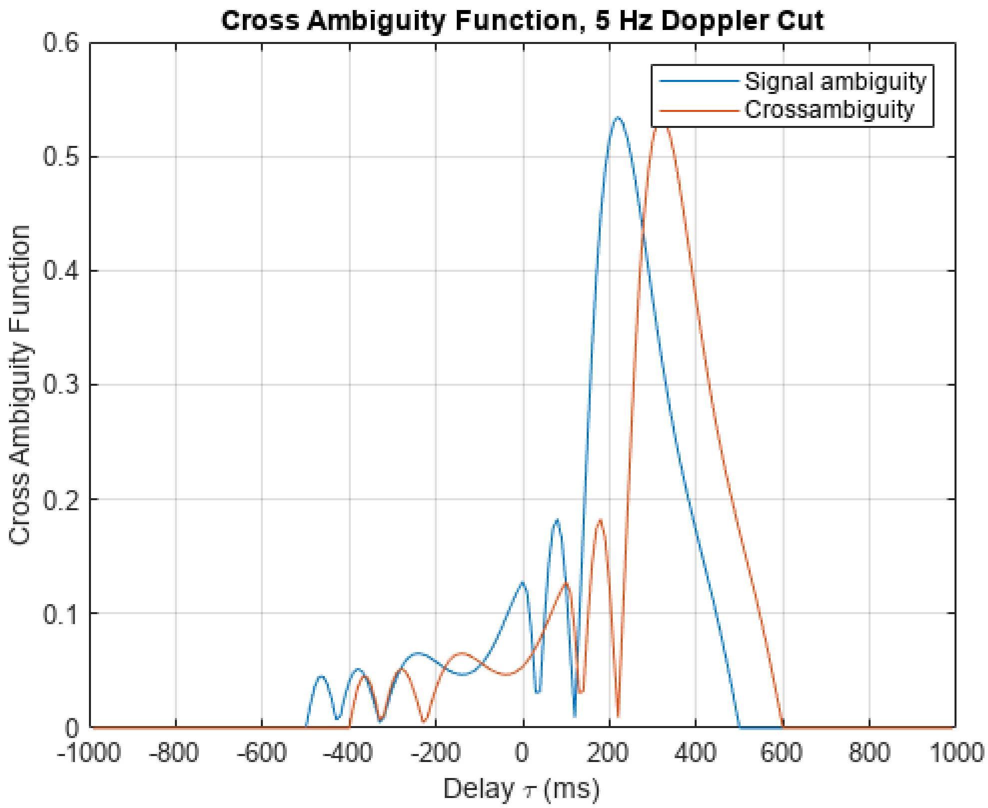
Create the original waveform and its delayed replica.

```
waveform1 = phased.LinearFMWaveform("SampleRate",fs,"PulseWidth",1,...
    "SweepBandwidth",bw1,"SweepDirection","Up","PulseWidth",pw,"PRF",prf);
wav1 = waveform1();
wav2 = [zeros(nlag,1);wav1(1:(end-nlag))];
```

Plot the ambiguity and crossambiguity functions.

```
ambgfun(wav1,fs,prf,"Cut","Doppler","CutVal",5)
hold on
```

```
ambgfun(wav1,wav2,fs,[prf,prf],"Cut","Doppler","CutVal",5)
hold off
legend("Signal ambiguity","Crossambiguity")
```



Input Arguments

x — Input pulse waveform

complex-valued row or column vector

Input pulse waveform, specified as a complex-valued row or column vector.

Example: `wf = phased.LinearFMWaveform; wf()`

y — Second input pulse waveform

complex-valued row or column vector

Second input pulse waveform, specified as a complex-valued row or column vector.

Example: `wf = phased.RectangularWaveform; wf()`

Fs — Sample rate

positive scalar

Sample rate in hertz, specified as a positive scalar.

Example: `wf = phased.LinearFMWaveform; wf.SampleRate`

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency in hertz, specified as a positive scalar.

Example: `wf = phased.LinearFMWaveform; wf.PRF`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `"Cut","Doppler","CutValue",10` specifies that a vector of ambiguity function values be produced at a Doppler shift of 10 Hz.

`Cut` — Direction of one-dimensional cut through ambiguity function
`"2D"` (default) | `"Delay"` | `"Doppler"`

Used to generate an ambiguity surface or one-dimensional cut through the ambiguity diagram. The value `"2D"` generates a surface plot of the two-dimensional ambiguity function. The direction of the one-dimensional cut is determined by setting the value of `"Cut"` to `"Delay"` or `"Doppler"`.

The choice of `"Delay"` generates a cut at zero time delay. In this case, the second output argument of `ambgfun`contains the ambiguity function values at Doppler shifted values. You can create a cut at nonzero time delay using the name-value pair `"CutValue"`.

The choice of `"Doppler"` generates a cut at zero Doppler shift. In this case, the second output argument of `ambgfun` contains the ambiguity function values at time-delayed values. You can create cut at nonzero Doppler using the name-value pair `"CutValue"`.

Data Types: `char` | `string`

`CutValue` — Optional time delay or Doppler shift at which ambiguity function cut is taken
0 (default) | real-valued scalar

When setting the name-value pair `"Cut"` to `"Delay"` or `"Doppler"`, you can set `"CutValue"` to specify a cross-section that may not coincide with either zero time delay or zero Doppler shift. However, `"CutValue"` cannot be used when `"Cut"` is set to `"2D"`.

When `"Cut"` is set to `"Delay"` ,`"CutValue"` is the time delay at which the cut is taken. Time delay units are in seconds.

When `"Cut"` is set to `"Doppler"`, `"CutValue"` is the Doppler shift at which the cut is taken. Doppler units are in hertz.

Example: `"CutValue",10.0`

Data Types: `double`

Output Arguments

`afmag` — Normalized ambiguity or crossambiguity function magnitudes
matrix

Normalized ambiguity or crossambiguity function magnitudes, returned as a matrix. `afmag` is an *M*-by-*N* matrix where *M* is the number of Doppler frequencies and *N* is the number of time delays.

`delay` — Time delays
vector

Time delays, returned as a vector. `delay` is an *N*-by-1 vector of time delays.

For the ambiguity function, if N_x is the length of signal `x`, then the delay vector consist of $N = 2N_x - 1$ samples in the range, $-(N_x/2) - 1,\ldots,(N_x/2) - 1)$.

For the crossambiguity function, let N_y be the length of the second signal. The time delay vector consists of $N = N_x + N_y - 1$ equally spaced samples. For an even number of delays, the delay sample times are $-(N/2 - 1)/Fs,\ldots,(N/2 - 1))/Fs$. For an odd number of delays, if $N_f = floor(N/2)$, the delay sample times are $-N_f/Fs,\ldots,(N_f - 1)/Fs$.

`doppler` — Doppler frequencies
vector

Doppler frequencies, returned as a vector. `doppler` is an *M*-by-1 vector of Doppler frequencies. The Doppler frequency vector consists of $M = 2^{ceil(log2\ N)}$ equally-spaced samples. Frequencies are $(-(M/2)F_s,\ldots,(M/2-1)F_s)$.

More About

Normalized Ambiguity Function

The normalized ambiguity function is

$$A(t, f_d) = \frac{1}{E_x}\left|\int_{-\infty}^{\infty} x(u)e^{j2\pi f_d u}x^*(u - t)\,du\right|$$

$$E_x = \int_{-\infty}^{\infty} x(u)x^*(u)\,du$$

where E_x is the squared norm of the signal, $x(t)$, t is the time delay, and f_d is the Doppler shift. The asterisk (*) denotes the complex conjugate. The ambiguity function describes the effects of time delays and Doppler shifts on the output of a matched filter.

The magnitude of the ambiguity function achieves maximum value at (0,0). At this point, there is perfect correspondence between the received waveform and the matched filter. The maximum value of the normalized ambiguity function is one.

The magnitude of the ambiguity function at zero time delay and Doppler shift, $|A(0, 0)|$, is the matched filter output when the received waveform exhibits the time delay and Doppler shift for which the matched filter is designed. Nonzero values of the time delay and Doppler shift variables indicate that the received waveform exhibits mismatches in time delay and Doppler shift from the matched filter.

The crossambiguity function between two different signals is

$$A(t, f_d) = \frac{1}{\sqrt{E_x E_y}} \left| \int_{-\infty}^{\infty} x(u) e^{j2\pi f_d u} y^*(u - t)\, du \right|$$

$$E_x = \int_{-\infty}^{\infty} x(u) x^*(u)\, du$$

$$E_x = \int_{-\infty}^{\infty} y(u) y^*(u)\, du$$

The peak of the crossambiguity function is not necessarily unity.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.
- Supported only when output arguments are specified.

References

[1] Levanon, N. and E. Mozeson. *Radar Signals*. Hoboken, NJ: John Wiley & Sons, 2004.

[2] Mahafza, B. R., and A. Z. Elsherbeni. *MATLAB Simulations for Radar Systems Design*. Boca Raton, FL: CRC Press, 2004.

[3] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

See Also

Functions
`pambgfun`

Objects
`phased.LinearFMWaveform` | `phased.MatchedFilter` | `phased.PhaseCodedWaveform` | `phased.RectangularWaveform` | `phased.SteppedFMWaveform`

aperture2gain

Convert effective aperture to gain

Syntax

```
GdB = aperture2gain(A,lambda)
```

Description

`GdB = aperture2gain(A,lambda)` returns the antenna gain `GdB` corresponding to an effective aperture `A` for an incident electromagnetic wave with wavelength `lambda`.

Examples

Compute Antenna Gain

An antenna has an effective aperture of 3 square meters. Find the antenna gain when used to capture an electromagnetic wave with a wavelength of 10 cm.

```
g = aperture2gain(3,0.1)
```

```
g = 35.7633
```

Input Arguments

`A` — Antenna effective aperture
positive scalar | *N*-element vector of positive values

Antenna effective aperture, specified as a positive scalar or as an *N*-element real-valued vector of positive values. If `A` is a vector, each element of `A` corresponds is the effective aperture of a different antenna. See “Gain and Effective Aperture” on page 2-20 for a discussion of aperture and gain. Units are in square meters.

Data Types: `double`

`lambda` — Wavelength of the incident electromagnetic wave
positive scalar

Wavelength of the incident electromagnetic wave, specified as a positive scalar. The same wavelength applies to all antennas in `A`. The wavelength of an electromagnetic wave is the ratio of the wave propagation speed to the frequency. Units are in meters.

Data Types: `double`

Output Arguments

`GdB` — Antenna gain
scalar | *N*-element real-valued vector

Antenna gain, returned as a scalar or as an N-element real-valued vector. The elements of `GdB` represent the gain corresponding to the elements in `A`. The size of `GdB` equals the size of `A`. Units are in dBi.

Data Types: `double`

More About

Gain and Effective Aperture

The effective aperture describes how much energy is captured by an antenna from an incident electromagnetic plane wave. The effective area of the antenna and is not the same as the actual physical area. The array gain of an antenna G is related to its effective aperture A_e by:

$$G = \frac{4\pi}{\lambda^2}A_e$$

where λ is the wavelength of the incident electromagnetic wave. For a fixed wavelength, the antenna gain is proportional to the effective aperture. For a fixed effective aperture, the antenna gain is inversely proportional to the square of the wavelength.

The gain expressed in dBi (GdB) is

$$GdB = 10\log_{10}G = 10\log_{10}(\frac{4\pi A_g}{\lambda^2}).$$

The effective antenna aperture can be derived from the gain in dB using

$$A_e = 10^{GdB/10}\frac{\lambda^2}{4\pi}.$$

Version History

Introduced in R2011a

References

[1] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

[2] Richards, M. *Fundamentals of Radar Signal Processing*, New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`gain2aperture`

arrayfactor

Array factor of sensor arrays

Syntax

```
af = arrayfactor(pos,ang)
af = arrayfactor(pos,ang,wts)
```

Description

`af = arrayfactor(pos,ang)` returns the array factor `af` for an array of elements located at positions `pos` and arrival directions `ang`.

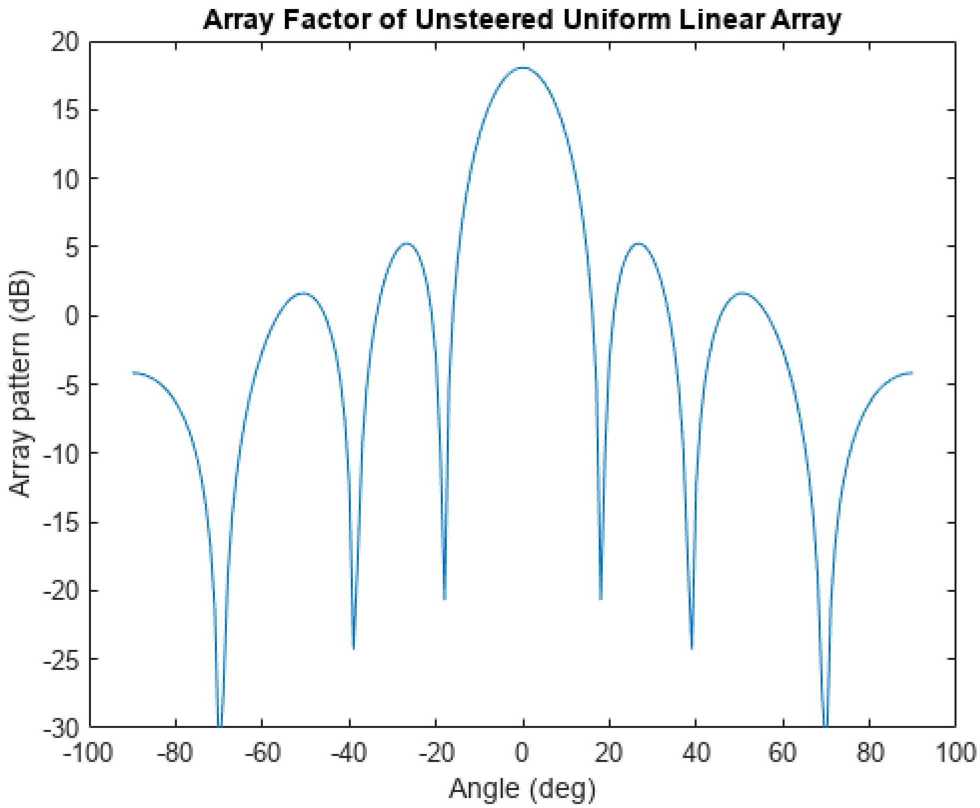
`af = arrayfactor(pos,ang,wts)` also specifies array steering weights `wts`.

Examples

Compute Array Factor of Unsteered ULA

Compute the array factor for an unsteered 8-element ULA operating at 100 MHz. Elements are placed 0.4 wavelengths apart.

```
freq = 100e6;
c = physconst('LightSpeed');
lam = c/freq;
pos = (0:7)*0.4;
ang = -90:90;
af = arrayfactor(pos,ang);
plot(ang,mag2db(abs(af.')))
xlabel('Angle (deg)')
ylabel('Array pattern (dB)')
ylim([-30 20])
title('Array Factor of Unsteered Uniform Linear Array')
```



Display Beam Patterns of Array Factor

Display the array beam patterns for an 8-element ULA with half-wavelength spacing. Steer the array to -30, 0, and 30 degrees.

Set the positions of the ULA elements.

```
pos = (0:7)*0.5;
```

Define the steering direction weights.

```
angsteer = [-30 0 30];
wsteer = steervec(pos,angsteer);
```

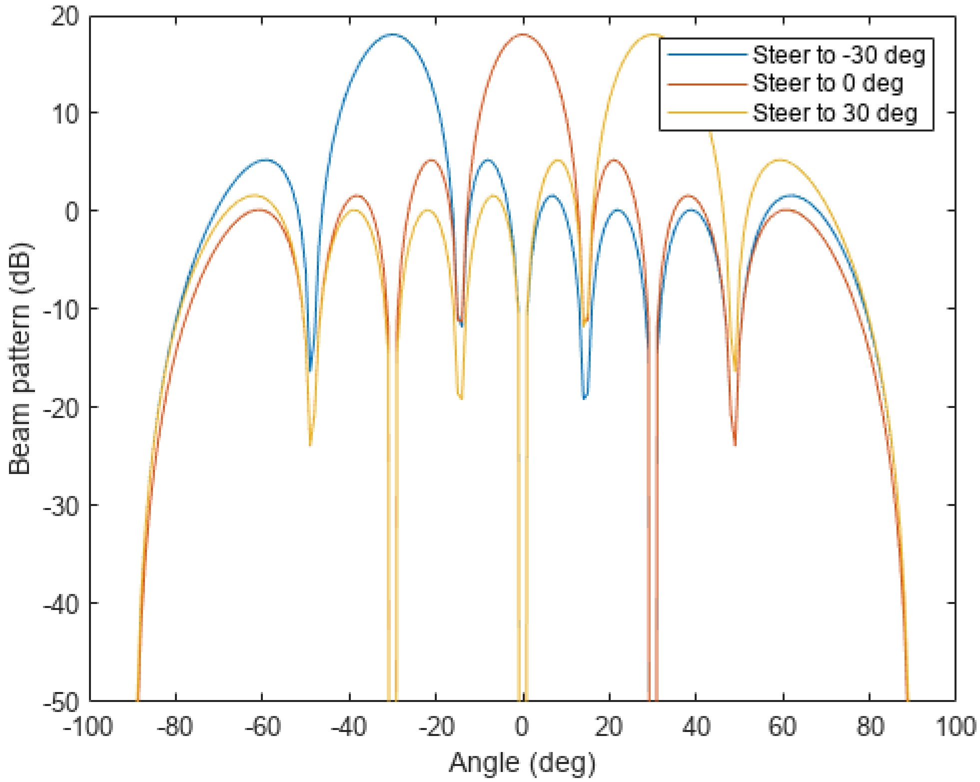
Compute the array factors for each steering direction for all angles.

```
ang = -90:90;
af = arrayfactor(pos,ang,wsteer);
```

Plot the array factor for each steering direction.

```
plot(ang,mag2db(abs(af.')))
xlabel('Angle (deg)')
ylabel('Beam pattern (dB)')
```

```
ylim([-50 20])
legend('Steer to -30 deg','Steer to 0 deg','Steer to 30 deg')
```



Input Arguments

pos — Positions of array sensor elements
1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array, specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero. When `pos` is a 2-by-*N* matrix, it represents the *(y,z)*-coordinates of the sensor elements of a planar array. This array is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. When `pos` is a 3-by-*N* matrix, then the array can have an arbitrary shape. Sensor positions are in terms of signal wavelength.

Example: `[0,0,0; 0.1,0.4,0.3; 1,1,1]`

Data Types: `double`

ang — Arrival directions of incoming signals
1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Arrival directions of incoming signals specified as a 1-by-*M* vector or a 2-by-*M* matrix, where *M* is the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the incoming signal `[az;el]`. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the arrival direction vector onto the *xy* plane. It is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the arrival direction vector and *xy*-plane. It is positive when measured towards the *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero. Angle units are specified in degrees.

Example: [45;0]

Data Types: `double`

`wts` — Array weights
`ones(N,1)` (default) | complex-valued *N*-by-1 vector | complex-valued *N*-by-*L* matrix

Array weights, specified as a complex-valued *N*-by-1 vector or complex-valued *N*-by-*L* matrix. Each column of `wts` corresponds to a different set of weights for the array. *L* is the number of weight sets.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`af` — Array factor of sensor array
1-by-*M* complex-valued vector | *L*-by-*M* complex-valued matrix

Array factor of sensor array, returned as a 1-by-*M* complex-valued vector or *L*-by-*M* complex-valued matrix. Each column contains the array factor for a different set of weights `wts`. Each row corresponds to one of the angles specified in `ang`.

Version History

Introduced in R2022a

References

[1] Van Trees, Harry L. *Detection, Estimation, and Modulation Theory. 4: Optimum Array Processing*. Wiley, 2002.

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`steervec` | `phased.SteeringVector`

az2broadside

Convert azimuth and elevation angle to broadside angle

Syntax

```
bsang = az2broadside(az)
bsang = az2broadside(az,el)
```

Description

`bsang = az2broadside(az)` returns the broadside angle on page 2-27, `bsang`, corresponding to the azimuth angle, `az`, and zero elevation angle. All angles are define with respect to the local coordinate system.

`bsang = az2broadside(az,el)` also specifies the elevation angle, `el`.

Examples

Convert Azimuth Angle to Broadside Angle

Return the broadside angle corresponding to 45° azimuth and 0° elevation.

```
bsang = az2broadside(45)
```

```
bsang = 45.0000
```

Convert Azimuth and Elevation to Broadside Angle

Return the broadside angle corresponding to 45° azimuth and 45° elevation.

```
bsang = az2broadside(45,45)
```

```
bsang = 30.0000
```

Convert Multiple Azimuth and Elevation Angles to Broadside Angles

Return broadside angles for 10 azimuth-elevation pairs.

```
az = (75:5:120)';
el = (45:5:90)';
bsang = az2broadside(az,el);
```

Input Arguments

az — Azimuth angle
scalar | vector of real values

Azimuth angle, specified as a scalar or vector of real values. Azimuth angles lie in the range from –180° to 180°. Units are in degrees.

Example: `[35;20;-10]`

el — Elevation angle
`0` (default) | scalar | vector of real values

Elevation angle, specified as a scalar or vector. The elevation angle lie in the range from –90° to 90°. The length of `el` must equal the length of `az`. Units are in degrees.

Example: `[5;2;-1]`

Output Arguments

bsang — Broadside angle
scalar | vector of real values

Broadside angle, returned as a scalar or vector. The length of `bsang` equals the length of `az`. Units are in degrees.

More About

Broadside Angle

Broadside angles are useful in describing the response pattern of a uniform linear array (ULA).

For the definition of the broadside angle and how to convert between azimuth and elevation, and broadside angle see “Broadside Angles”. For definitions of the azimuth and elevation angles, see “Azimuth and Elevation Angles”.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`broadside2az` | `azel2uv` | `azel2phitheta`

azel2phitheta

Convert angles from azimuth-elevation form to phi-theta form

Syntax

```
PhiTheta = azel2phitheta(AzEl)
PhiTheta = azel2phitheta(AzEl,RotAx)
```

Description

`PhiTheta = azel2phitheta(AzEl)` converts the azimuth/elevation angle on page 2-29 pairs to their corresponding phi/theta angle on page 2-30 pairs.

`PhiTheta = azel2phitheta(AzEl,RotAx)` also specifies the choice of phi-theta angle convention using `RotAx`.

Examples

Convert Azimuth-Elevation Coordinates to Phi-Theta Coordinates

Find the phi-theta representation for 30° azimuth and 10° elevation for the convention where phi is defined from the *y*-axis to the *z*-axis, and theta is defined from the *x*-axis toward the *yz*-plane.

```
PhiTheta = azel2phitheta([30;10])
```

```
PhiTheta = 2×1

   19.4254
   31.4749
```

Azimuth-Elevation Coordinates to Alternative Phi-Theta Coordinates

Find the phi-theta representation for 30° azimuth and 10° elevation for the convention with phi defined from the *x*-axis to the *y*-axis, and theta defined from the *z*-axis toward the *xy*-plane.

```
PhiTheta = azel2phitheta([30;10],false)
```

```
PhiTheta = 2×1

    30
    80
```

Input Arguments

`AzEl` — Azimuth-elevation angle pairs
two-row matrix

Azimuth and elevation angles, specified as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [azimuth; elevation].

Data Types: `double`

`RotAx` — Phi-theta angle convention selection
`true` (default) | `false`

Phi-theta angle convention selection, specified as `true` or `false`.

- If `RotAx` is `true`, the phi angle is defined from the *y*-axis to the *z*-axis and the theta angle is defined from the *x*-axis toward the *yz*-plane.
- If `RotAx` is `false`, the phi angle is defined from the *x*-axis to the *y*-axis and the theta angle is defined from the *z*-axis toward the *xy*- plane. (see "Alternative Definition of Phi and Theta" on page 2-31).

Data Types: `double`

Output Arguments

`PhiTheta` — Phi-theta angle pairs
two-row matrix

Phi and theta angles, returned as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [phi; theta]. The matrix dimensions of `PhiTheta` are the same as those of `AzEl`.

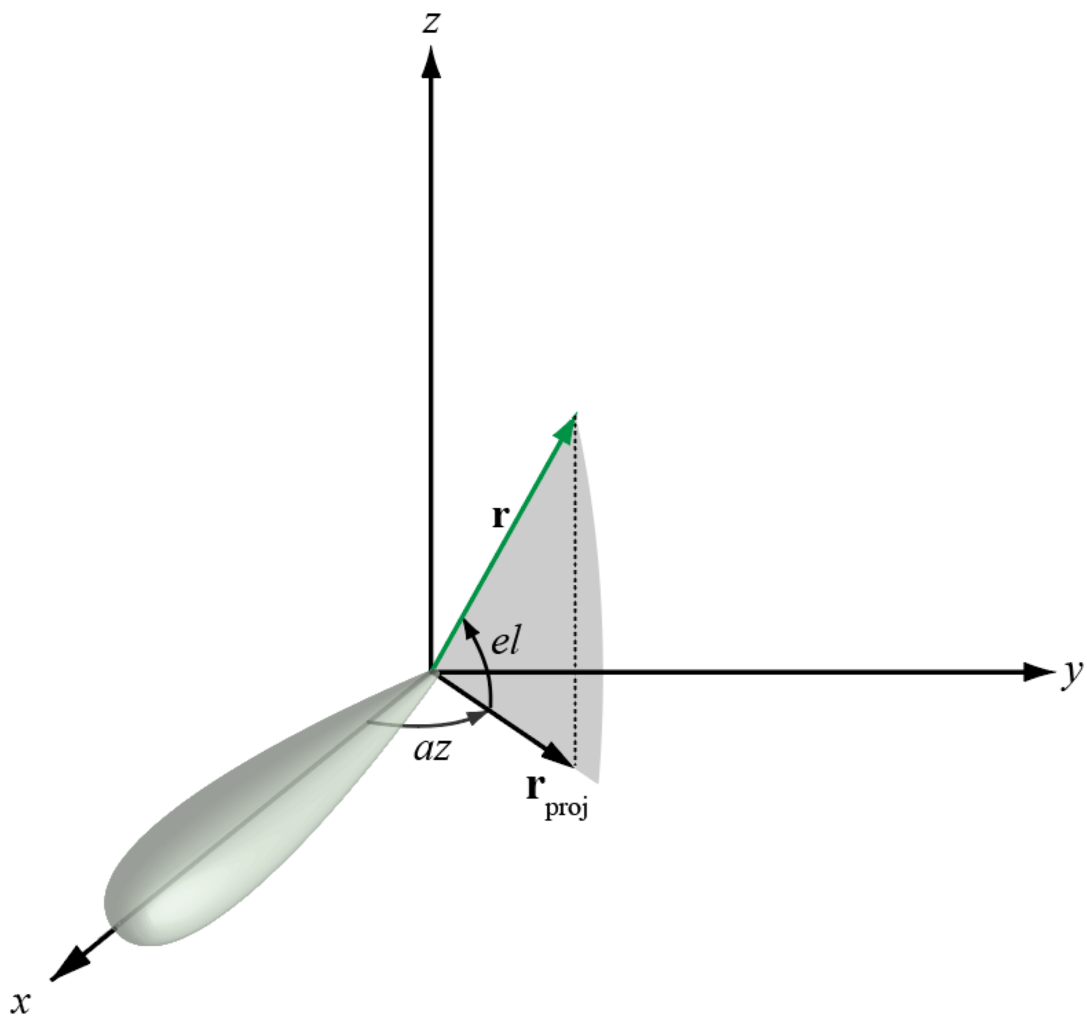
More About

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

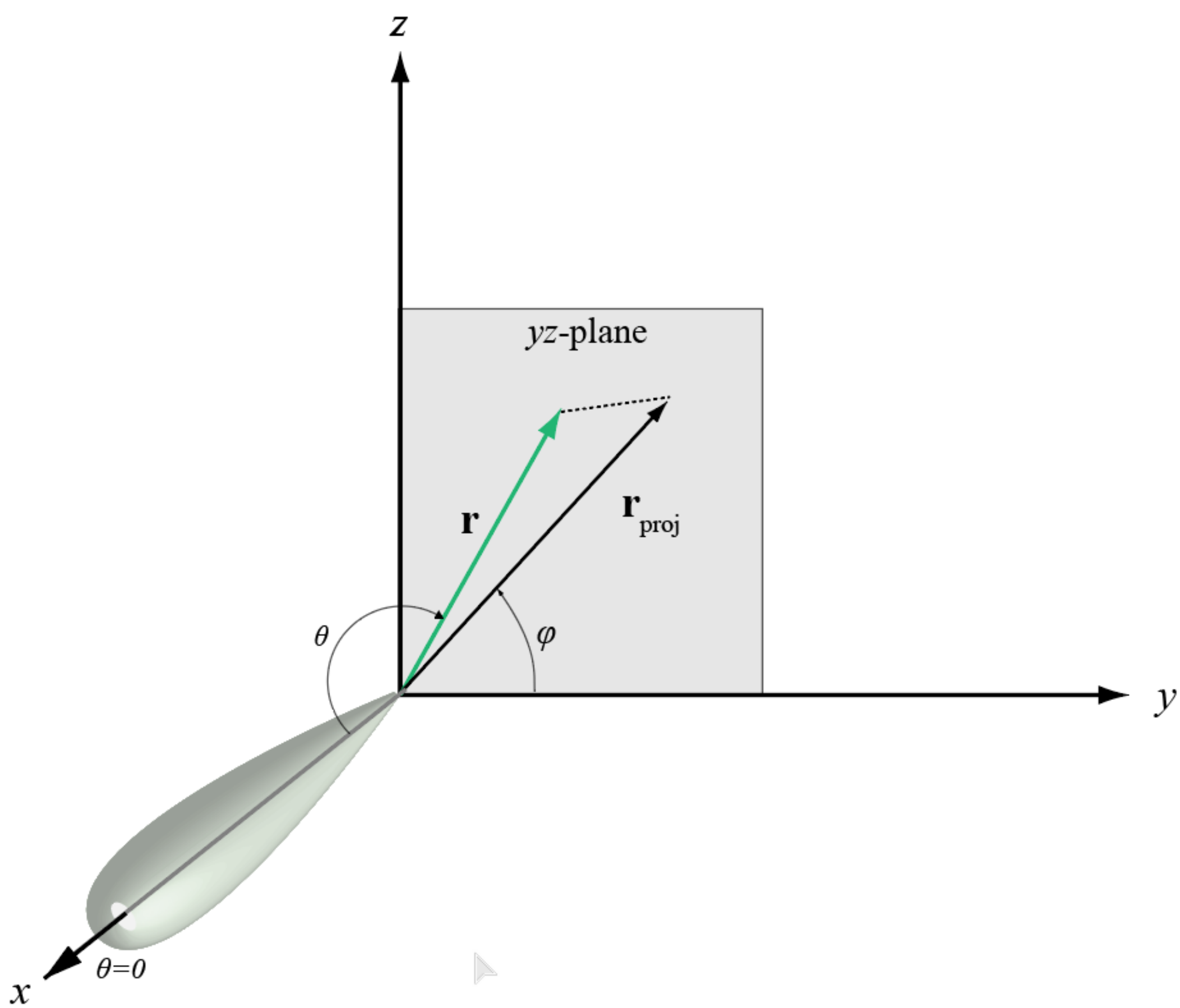
This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



Phi and Theta Angles

The phi angle (φ) is the angle from the positive *y*-axis to the vector's orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi\sin\theta$

$\tan az = \cos\phi\tan\theta$

$\cos\theta = \cos el\cos az$

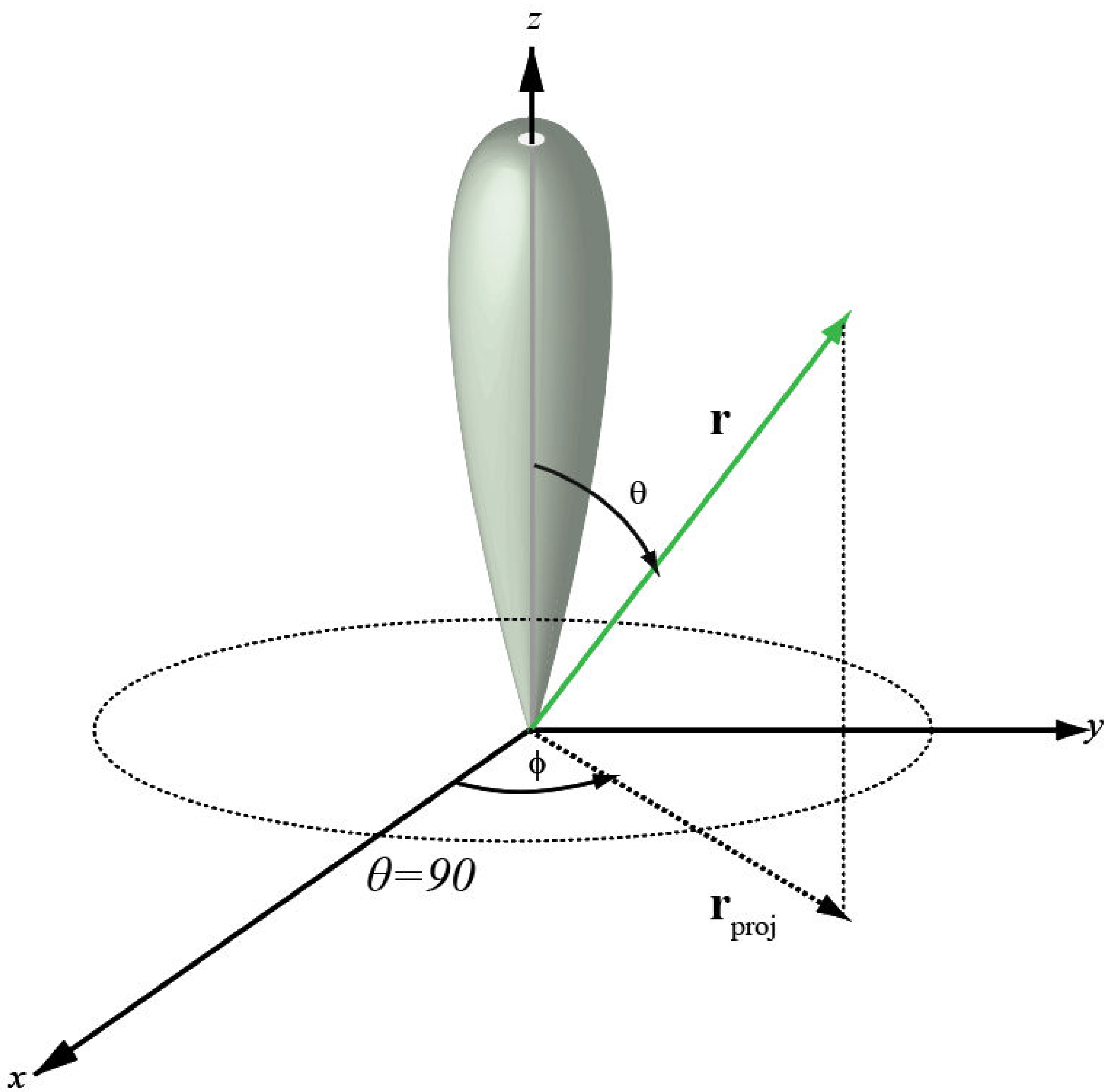
$\tan\phi = \tan el/\sin az$

This transformation applies when `RotAx` is `true`.

Alternative Definition of Phi and Theta

The phi angle (φ) is the angle from the positive *x*-axis to the vector's orthogonal projection onto the *xy* plane. The angle is positive toward the positive *y*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *z*-axis to the vector itself. The angle is positive toward the *xy* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line.



$\phi = az$
$\theta = 90 - el$
$az = \phi$
$el = 90 - \theta$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phitheta2azel`

Topics

"Spherical Coordinates"

azel2phithetapat

Convert radiation pattern from azimuth-elevation coordinates to phi-theta coordinates

Syntax

```
pat_phitheta = azel2phithetapat(pat_azel,az,el)
pat_phitheta = azel2phithetapat(pat_azel,az,el,phi,theta)
pat_phitheta = azel2phithetapat( ___ ,'RotateZ2X',rotpatax)
[pat_phitheta,phi_pat,theta_pat] = azel2phithetapat( ___ )
```

Description

`pat_phitheta = azel2phithetapat(pat_azel,az,el)` converts the antenna radiation pattern, `pat_azel`, from azimuth and elevation coordinates to the pattern, `pat_phitheta`, in phi and theta coordinates. `az` and `el` are the azimuth and elevation angles at which the `pat_azel` values are defined. The `pat_phitheta` matrix covers theta values from 0 to 180 degrees and phi values from 0 to 360 degrees in one degree increments. The function interpolates the `pat_azel` matrix to estimate the response of the antenna in a given phi-theta direction.

`pat_phitheta = azel2phithetapat(pat_azel,az,el,phi,theta)` also specifies `phi` and `theta` as the grid at which to sample `pat_phitheta`. To avoid interpolation errors, `phi` should cover the range [0,180], and `theta` should cover the range [0,360].

`pat_phitheta = azel2phithetapat( ___ ,'RotateZ2X',rotpatax)` also specifies `rotpatax` to indicate the boresight direction of the pattern along the *x*-axis or the *z*-axis.

`[pat_phitheta,phi_pat,theta_pat] = azel2phithetapat( ___ )` also returns vectors `phi_pat` and `theta_pat` containing the phi and theta angles at which `pat_phitheta` is sampled.

Examples

Convert Radiation Pattern to Phi-Theta

Convert a radiation pattern to φ/θ form, with the φ and θ angles spaced 1 degree apart.

Define the pattern in terms of azimuth and elevation.

```
az = -180:180;
el = -90:90;
pat_azel = mag2db(repmat(cosd(el)',1,numel(az)));
```

Convert the pattern to φ/θ space.

```
pat_phitheta = azel2phithetapat(pat_azel,az,el);
```

Plot Converted Radiation Pattern

Plot the result of converting a radiation pattern to ϕ/θ space with the ϕ and θ angles spaced 1 degree apart.

The radiation pattern is the cosine of the elevation.

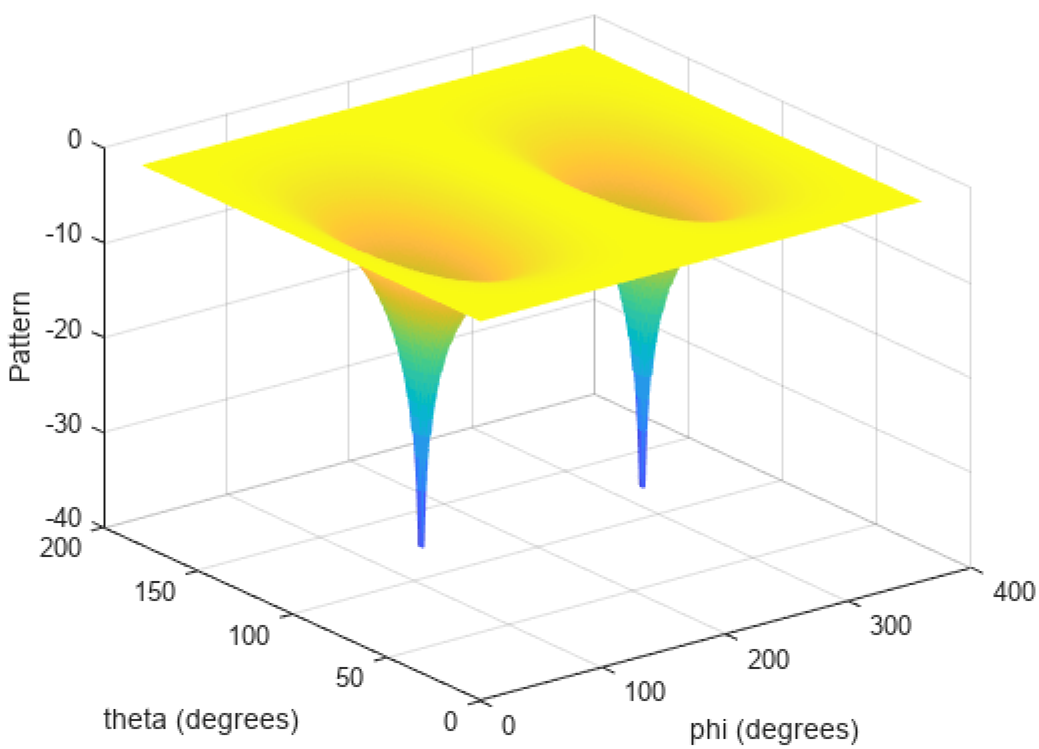
```
az = -180:180;
el = -90:90;
pat_azel = repmat(cosd(el)',1,numel(az));
```

Convert the pattern to ϕ/θ space. Use the returned ϕ and θ angles for plotting.

```
[pat_phitheta,phi,theta] = azel2phithetapat(pat_azel,az,el);
```

Plot the result.

```
H = surf(phi,theta,mag2db(pat_phitheta));
H.LineStyle = 'none';
xlabel('phi (degrees)');
ylabel('theta (degrees)');
zlabel('Pattern');
```

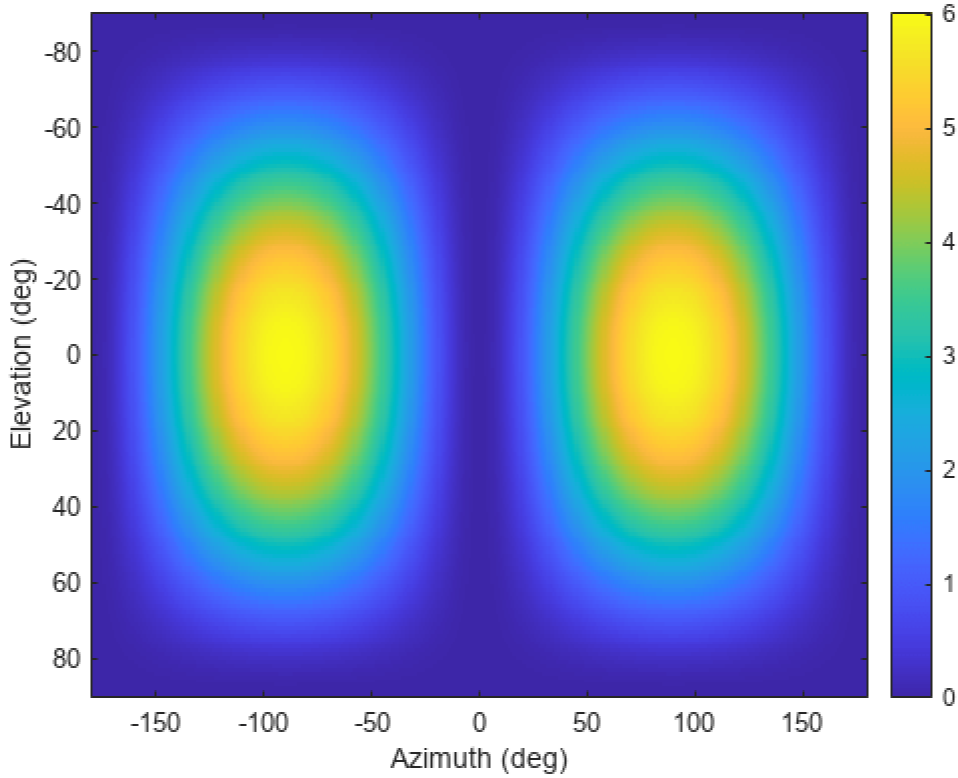


Convert Radiation Pattern to Alternate Phi-Theta Coordinates

Convert a radiation pattern to the alternate phi-theta coordinates, with the phi and theta angles spaced one degree apart.

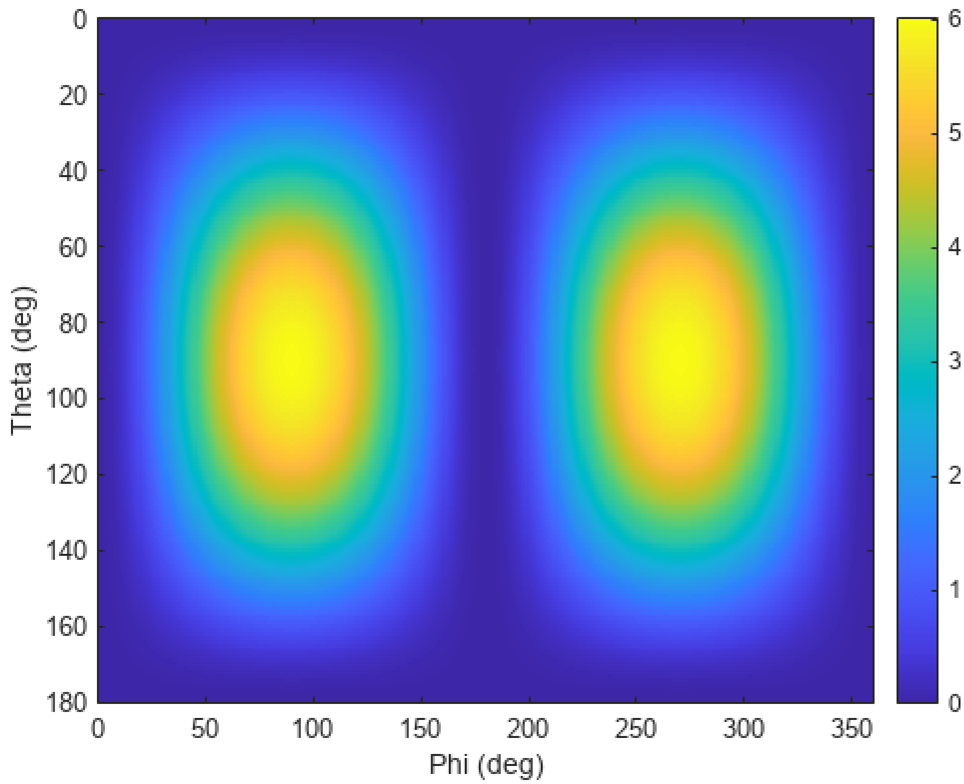
Create a simple radiation pattern in terms of azimuth and elevation. Add an offset to the pattern to suppress taking the logarithm of zero in `mag2db`.

```
az = -180:180;
el = -90:90;
pat_azel = mag2db(cosd(el).^2'*sind(az).^2 + 1);

imagesc(az,el,pat_azel)
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
colorbar
```



Convert the pattern to phi-theta space.

```
[pat_phitheta,phi_pat,theta_pat] = azel2phithetapat(pat_azel,az,el,'RotateZ2X',false);
imagesc(phi_pat,theta_pat,pat_phitheta)
xlabel('Phi (deg)')
ylabel('Theta (deg)')
colorbar
```



Convert Radiation Pattern for Specific Phi/Theta Values

Convert a radiation pattern to ϕ/θ space with ϕ and θ angles spaced 5 degrees apart.

The radiation pattern is the cosine of the elevation.

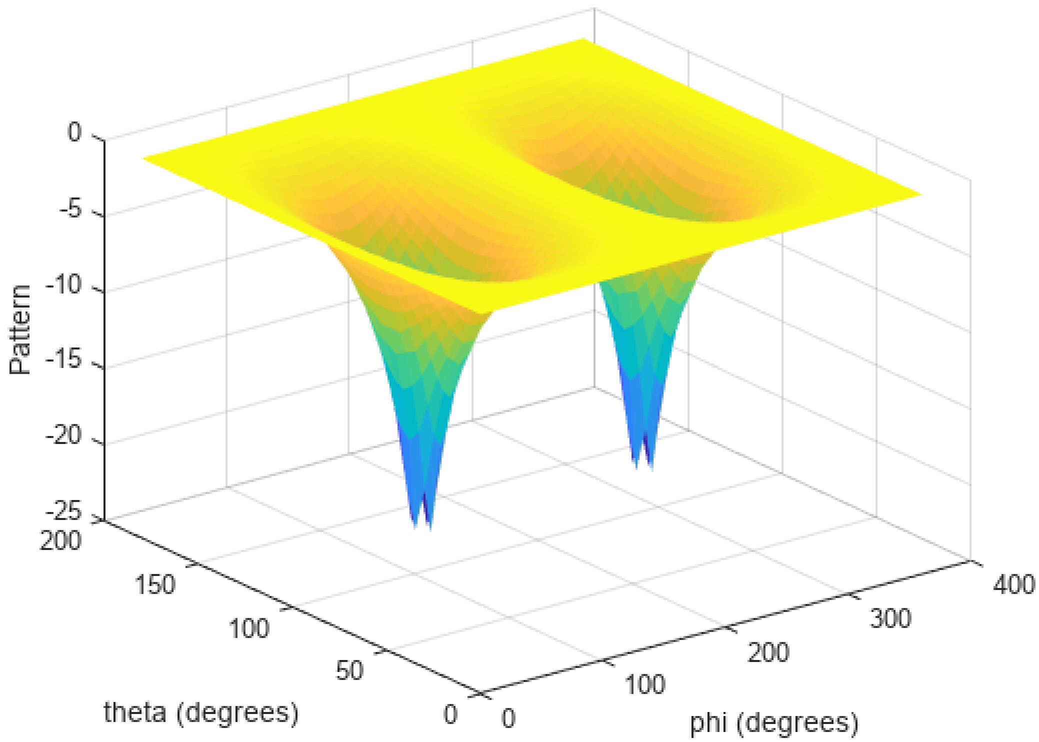
```
az = -180:180;
el = -90:90;
pat_azel = repmat(cosd(el)',1,numel(az));
```

Define the set of ϕ and θ angles at which to sample the pattern. Then, convert the pattern.

```
phi = 0:5:360;
theta = 0:5:180;
pat_phitheta = azel2phithetapat(pat_azel,az,el,phi,theta);
```

Plot the result.

```
H = surf(phi,theta,mag2db(pat_phitheta));
H.LineStyle = 'none';
xlabel('phi (degrees)');
ylabel('theta (degrees)');
zlabel('Pattern');
```



Input Arguments

pat_azel — Antenna radiation pattern
real-valued Q-by-P matrix

Antenna radiation pattern as a function of azimuth and elevation, specified as a real-valued Q-by-P matrix. `pat_azel` contains the magnitude pattern. P is the length of the `az` vector, and Q is the length of the `el` vector. Units are in dB.

Data Types: `double`

az — Azimuth angles
real-valued length-P vector

Azimuth angles at which the `pat_azel` pattern is sampled, specified as a real-valued length-P vector. Azimuth angles lie between –180 and 180, inclusive. Units are in degrees.

Data Types: `double`

el — Elevation angles
real-valued length-Q vector

Elevation angles at which the `pat_azel` pattern is sampled, specified as a real-valued length-Q vector. Azimuth angles lie between –90 and 90, inclusive. Units are in degrees.

Data Types: `double`

phi — Phi angles
[0:360] (default) | real-valued length-*L* vector

Phi angles at which the pat_phitheta pattern is sampled, specified as a real-valued length-*L* vector. Phi angles lie between 0 and 360, inclusive. Units are in degrees.

Data Types: double

theta — Theta angles
[0:180] (default) | real-valued length-*M* vector

Theta angles at which the pat_phitheta pattern is sampled, specified as a real-valued length-*M* vector. Theta angles lie between 0 and 180, inclusive. Units are in degrees.

Data Types: double

rotpatax — Pattern boresight direction selector
true (default) | false

Pattern boresight direction selector, specified as true or false.

- If rotpatax is true, the pattern boresight is along the *x*-axis. In this case, the *z*-axis of phi-theta space is aligned with the *x*-axis of azimuth and elevation space. The phi angle is defined from the *y*-axis to the *z*-axis and the theta angle is defined from the *x*-axis toward the *yz*-plane. (See “Phi and Theta Angles” on page 2-41).
- If rotpatax is false, the phi angle is defined from the *x*-axis to the *y*-axis and the theta angle is defined from the *z*-axis toward the *xy*-plane. (See “Alternative Definition of Phi and Theta” on page 2-42).

Data Types: logical

Output Arguments

pat_phitheta — Antenna radiation pattern in phi-theta coordinates
real-valued *M*-by-*L* matrix

Antenna radiation pattern in phi-theta coordinates, returned as a real-valued *M*-by-*L* matrix. pat_phitheta represents the magnitude pattern. *L* is the length of the phi_pat vector, and *M* is the length of the theta_pat vector. Units are in dB.

phi_pat — Phi angles
real-valued length-*L* vector

Phi angles at which the pat_phitheta pattern is sampled, returned as a real-valued length *L* vector. Units are in degrees.

theta_pat — Theta angles
real-valued length-*M* vector

Theta angles at which the pat_phitheta pattern is sampled, returned as a real-valued length-*M* vector. Units are in degrees.

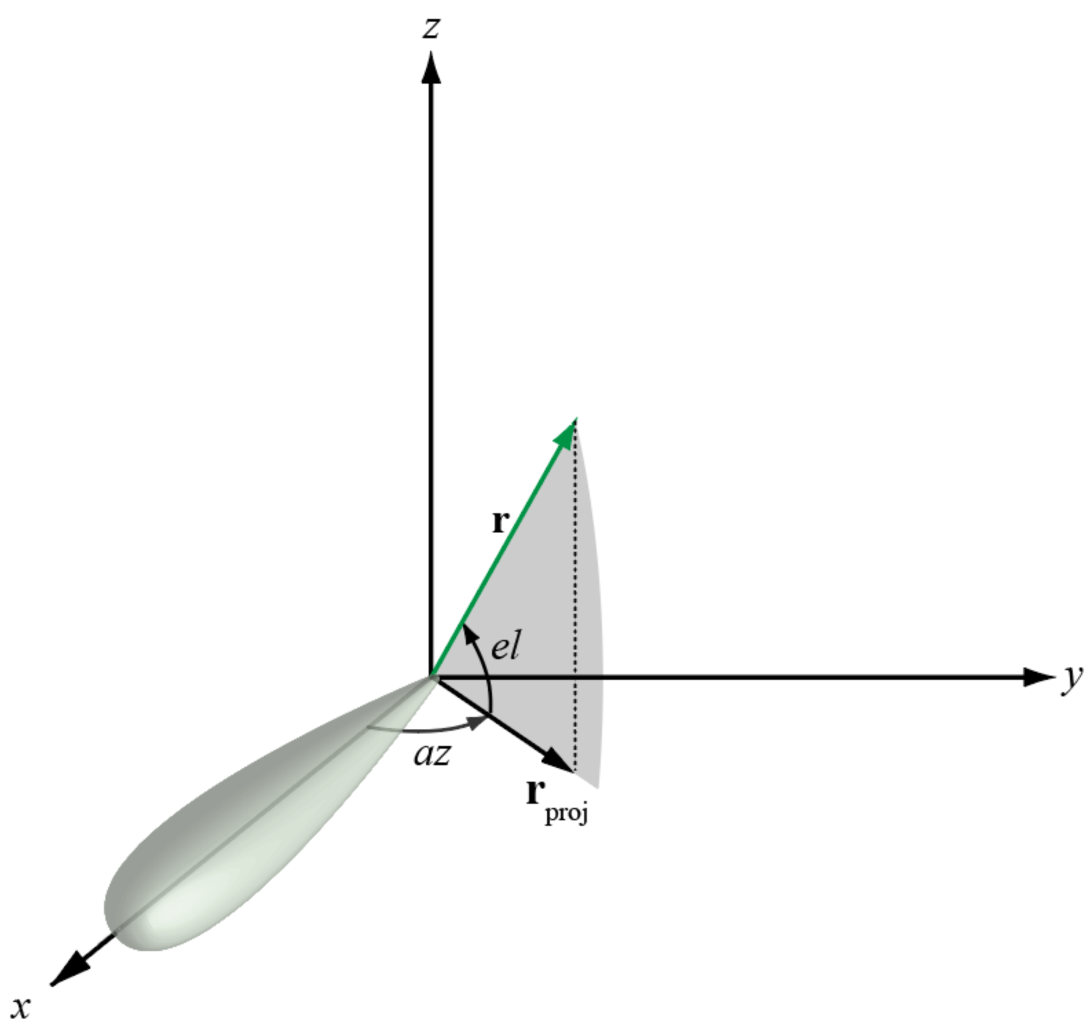
More About

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

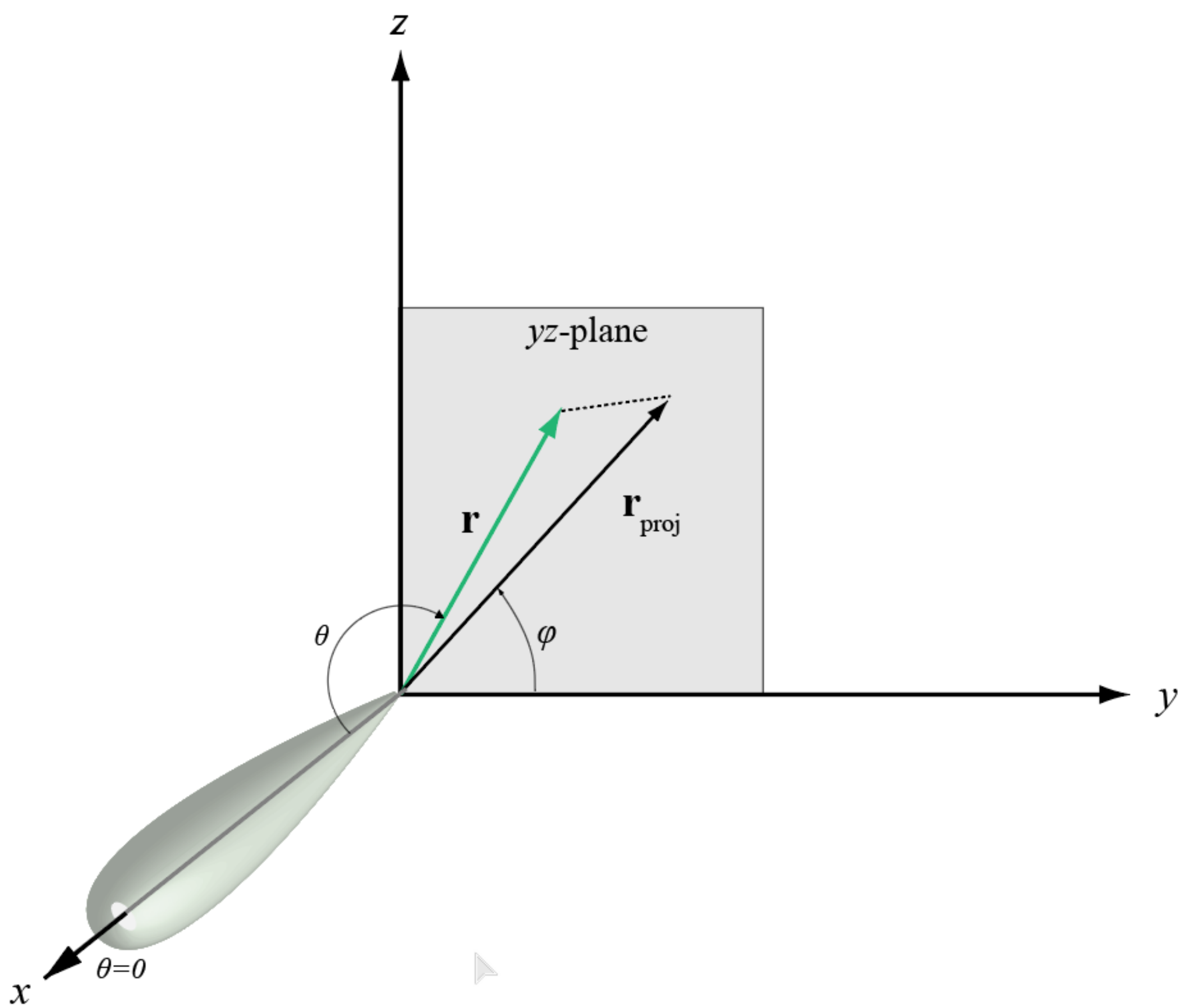
This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



Phi and Theta Angles

The phi angle (*φ*) is the angle from the positive *y*-axis to the vector's orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (*θ*) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi \sin\theta$

$\tan az = \cos\phi \tan\theta$

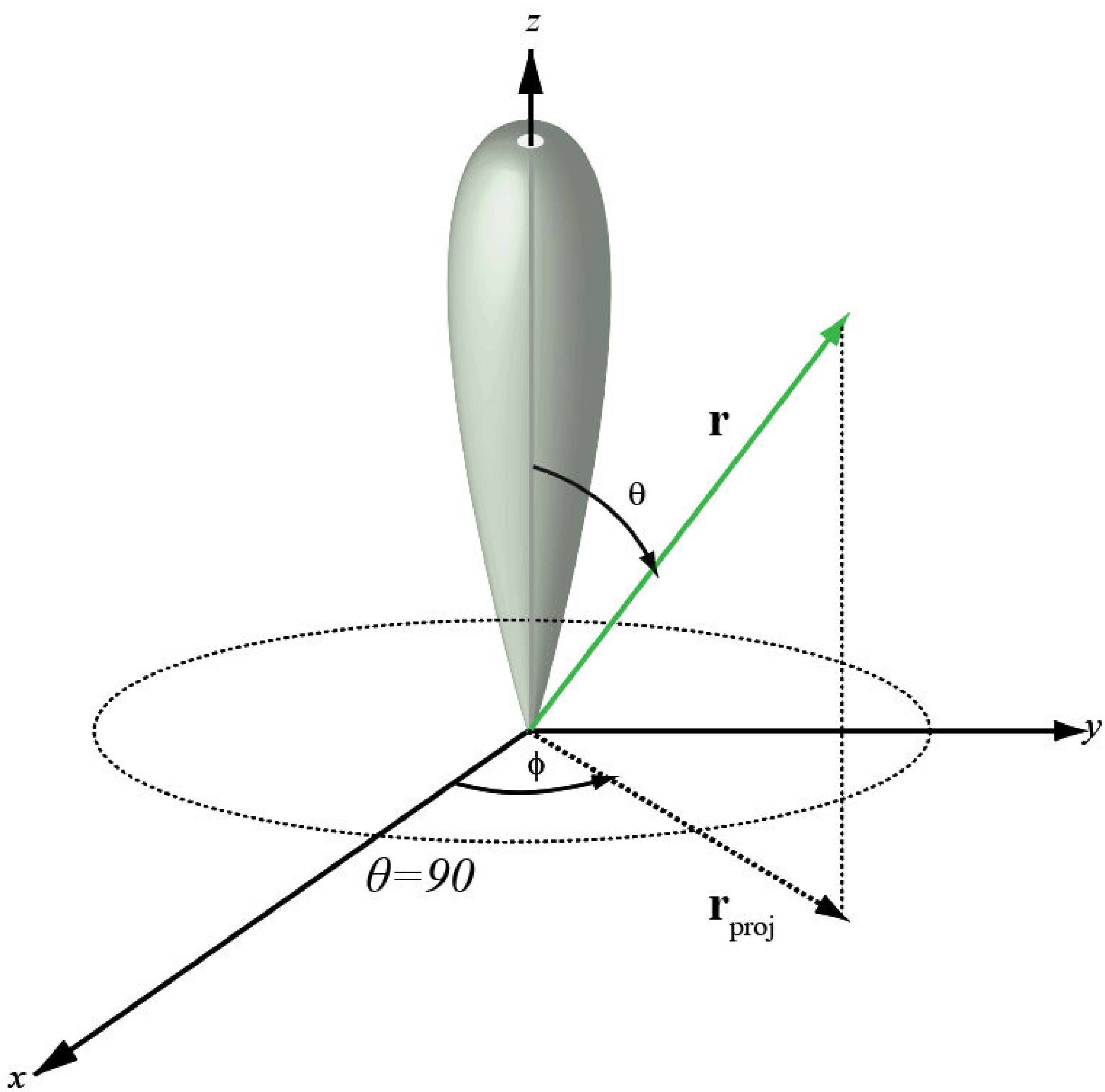
$\cos\theta = \cos el \cos az$

$\tan\phi = \tan el / \sin az$

Alternative Definition of Phi and Theta

The phi angle (φ) is the angle from the positive x-axis to the vector's orthogonal projection onto the xy plane. The angle is positive toward the positive y-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the z-axis to the vector itself. The angle is positive toward the xy plane. The theta angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line.



$$\phi = az$$
$$\theta = 90 - el$$
$$az = \phi$$
$$el = 90 - \theta$$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phased.CustomAntennaElement` | `phitheta2azel` | `azel2phitheta` | `phitheta2azelpat`

Topics

"Spherical Coordinates"

azel2uv

Convert azimuth/elevation angles to u/v coordinates

Syntax

```
UV = azel2uv(AzEl)
```

Description

`UV = azel2uv(AzEl)` converts the azimuth/elevation angle on page 2-45 pairs to their corresponding coordinates in *u*/*v* space on page 2-46.

Examples

Conversion of Azimuth and Elevation to UV

Find the corresponding *uv* representation for 30° azimuth and 0° elevation.

```
uv = azel2uv([30;0])
```

```
uv = 2×1

    0.5000
         0
```

Input Arguments

`AzEl` — Azimuth/elevation angle pairs

two-row matrix

Azimuth and elevation angles, specified as a two-row matrix. Each column of the matrix represents an angle pair in the form `[azimuth;elevation]`. Azimuth angles must lie in the range [*-90, 90*]. Units are in degrees.

Data Types: `double`

Output Arguments

UV — Angle in u/v space

two-row matrix

Angle in *u*/*v* space, returned as a two-row matrix. Each column of the matrix represents an angle in the form [*u*; *v*]. The matrix dimensions of `UV` are the same as those of `AzEl`.

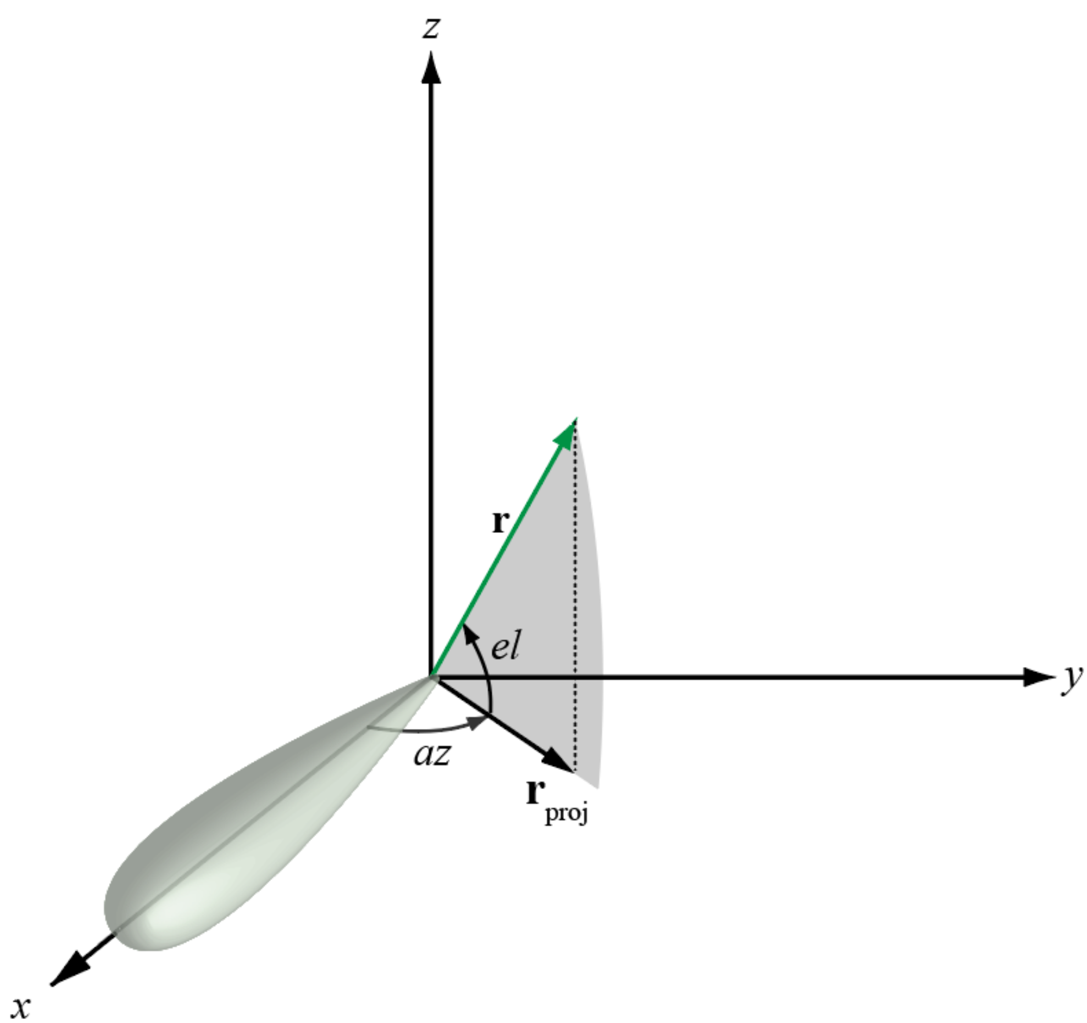
More About

Azimuth Angle, Elevation Angle

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



U/V Space

The *u/v* coordinates for the positive hemisphere $x \geq 0$ can be derived from the phi and theta angles on page 2-46.

The relation between these two coordinates systems is

$$u = \sin\theta\cos\phi$$
$$v = \sin\theta\sin\phi$$

In these expressions, φ and θ are the phi and theta angles, respectively.

To convert azimuth and elevation to *u* and *v* use the transformation

$$u = \cos el \sin az$$
$$v = \sin el$$

which is valid only in the range *abs(az)≤=90*.

The values of *u* and *v* satisfy the inequalities

$$-1 \leq u \leq 1$$
$$-1 \leq v \leq 1$$
$$u^2 + v^2 \leq 1$$

Conversely, the phi and theta angles can be written in terms of *u* and *v* using

$$\tan\phi = v/u$$
$$\sin\theta = \sqrt{u^2 + v^2}$$

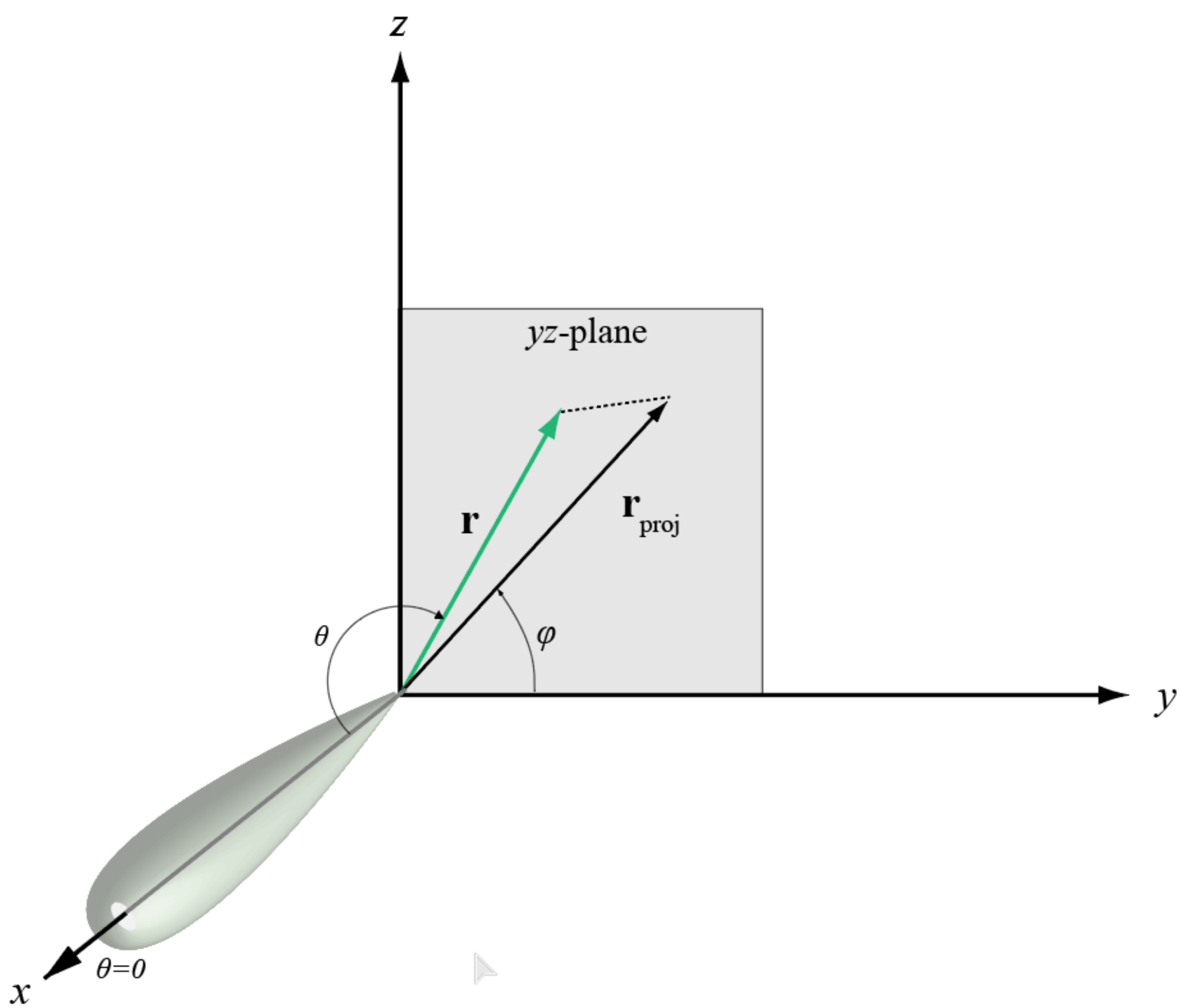
The azimuth and elevation angles can also be written in terms of *u* and *v*:

$$\sin el = v$$
$$\tan az = \frac{u}{\sqrt{1 - u^2 - v^2}}$$

Phi Angle, Theta Angle

The phi angle (φ) is the angle from the positive *y*-axis to the vector's orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi \sin\theta$

$\tan az = \cos\phi \tan\theta$

$\cos\theta = \cos el \cos az$

$\tan\phi = \tan el / \sin az$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`uv2azel`

Topics
"Spherical Coordinates"

azel2uvpat

Convert radiation pattern from azimuth/elevation form to u/v form

Syntax

```
pat_uv = azel2uvpat(pat_azel,az,el)
pat_uv = azel2uvpat(pat_azel,az,el,u,v)
[pat_uv,u_pat,v_pat] = azel2uvpat( ___ )
```

Description

`pat_uv = azel2uvpat(pat_azel,az,el)` expresses the antenna radiation pattern `pat_azel` in u/v space on page 2-54 coordinates instead of azimuth/elevation angle on page 2-53 coordinates. `pat_azel` samples the pattern at azimuth angles in `az` and elevation angles in `el`. The `pat_uv` matrix uses a default grid that covers *u* values from –1 to 1 and *v* values from –1 to 1. In this grid, `pat_uv` is uniformly sampled with a step size of 0.01 for *u* and *v*. The function interpolates to estimate the response of the antenna at a given direction. Values in `pat_uv` are NaN for *u* and *v* values outside the unit circle because *u* and *v* are undefined outside the unit circle.

`pat_uv = azel2uvpat(pat_azel,az,el,u,v)` uses vectors `u` and `v` to specify the grid at which to sample `pat_uv`. To avoid interpolation errors, `u` should cover the range [–1, 1] and `v` should cover the range [–1, 1].

`[pat_uv,u_pat,v_pat] = azel2uvpat( ___ )` returns vectors containing the *u* and *v* coordinates at which `pat_uv` samples the pattern, using any of the input arguments in the previous syntaxes.

Examples

Convert Radiation Pattern to UV Space

Convert a radiation pattern to *u*-*v* space, with the *u* and *v* coordinates spaced by 0.01.

Define the pattern in terms of azimuth and elevation.

```
az = -90:90;
el = -90:90;
pat_azel = mag2db(repmat(cosd(el)',1,numel(az)));
```

Convert the pattern to *u*-*v* space.

```
pat_uv = azel2uvpat(pat_azel,az,el);
```

Plot Converted Radiation Pattern

Plot the result of converting a radiation pattern to *u*/*v* space with the *u* and *v* coordinates spaced by 0.01.

The radiation pattern is the cosine of the elevation angle.

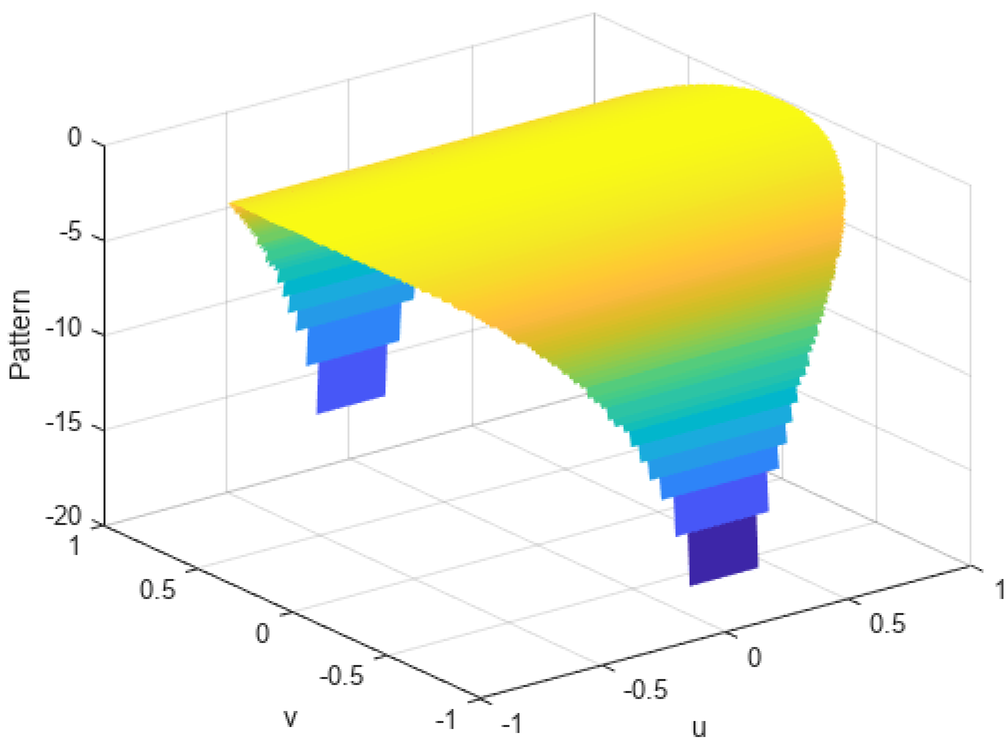
```
az = -90:90;
el = -90:90;
pat_azel = repmat(cosd(el)',1,numel(az));
```

Convert the pattern to *u*/*v* space. Use the *u* and *v* coordinates for plotting.

```
[pat_uv,u,v] = azel2uvpat(pat_azel,az,el);
```

Plot the result.

```
H = surf(u,v,mag2db(pat_uv));
H.LineStyle = 'none';
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



Convert Radiation Pattern for Specific U/V Values

Convert a radiation pattern to *u*/*v* form, with the *u* and *v* coordinates spaced by 0.05.

The radiation pattern is cosine of the elevation angle.

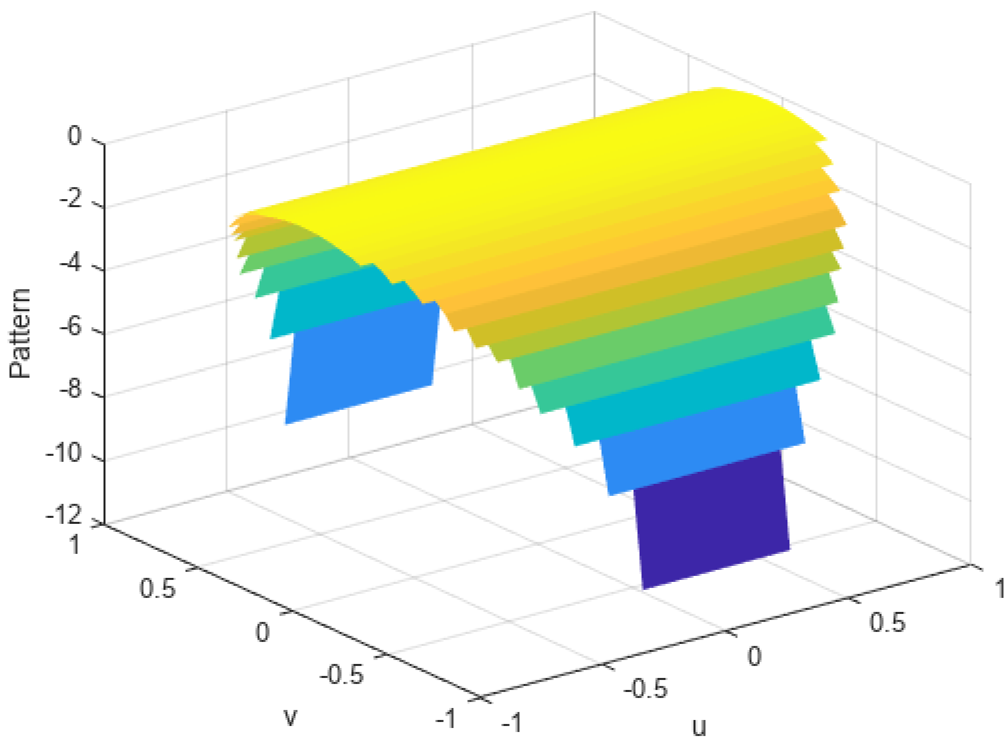
```
az = -90:90;
el = -90:90;
pat_azel = repmat(cosd(el)',1,numel(az));
```

Define the set of *u* and *v* coordinates at which to sample the pattern. Then, convert the pattern.

```
u = -1:0.05:1;
v = -1:0.05:1;
pat_uv = azel2uvpat(pat_azel,az,el,u,v);
```

Plot the result.

```
H = surf(u,v,mag2db(pat_uv));
H.LineStyle = 'none';
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



Input Arguments

pat_azel — Antenna radiation pattern in azimuth/elevation form
Q-by-P matrix

Antenna radiation pattern in azimuth/elevation form, specified as a Q-by-P matrix. `pat_azel` samples the 3-D magnitude pattern in decibels, in terms of azimuth and elevation angles. P is the length of the `az` vector, and Q is the length of the `el` vector.

Data Types: double

az — Azimuth angles
vector of length P

Azimuth angles at which pat_azel samples the pattern, specified as a vector of length P. Each azimuth angle is in degrees, between –90 and 90. Such azimuth angles are in the hemisphere for which *u* and *v* are defined.

Data Types: double

el — Elevation angles
vector of length Q

Elevation angles at which pat_azel samples the pattern, specified as a vector of length Q. Each elevation angle is in degrees, between –90 and 90.

Data Types: double

u — *u* coordinates
[-1:0.01:1] (default) | vector of length L

u coordinates at which pat_uv samples the pattern, specified as a vector of length L. Each *u* coordinate is between –1 and 1.

Data Types: double

v — *v* coordinates
[-1:0.01:1] (default) | vector of length M

v coordinates at which pat_uv samples the pattern, specified as a vector of length M. Each *v* coordinate is between –1 and 1.

Data Types: double

Output Arguments

pat_uv — Antenna radiation pattern in *u*/*v* form
M-by-L matrix

Antenna radiation pattern in *u*/*v* form, returned as an M-by-L matrix. pat_uv samples the 3-D magnitude pattern in decibels, in terms of *u* and *v* coordinates. L is the length of the u vector, and M is the length of the v vector. Values in pat_uv are NaN for *u* and *v* values outside the unit circle because *u* and *v* are undefined outside the unit circle.

u_pat — *u* coordinates
vector of length L

u coordinates at which pat_uv samples the pattern, returned as a vector of length L.

v_pat — *v* coordinates
vector of length M

v coordinates at which pat_uv samples the pattern, returned as a vector of length M.

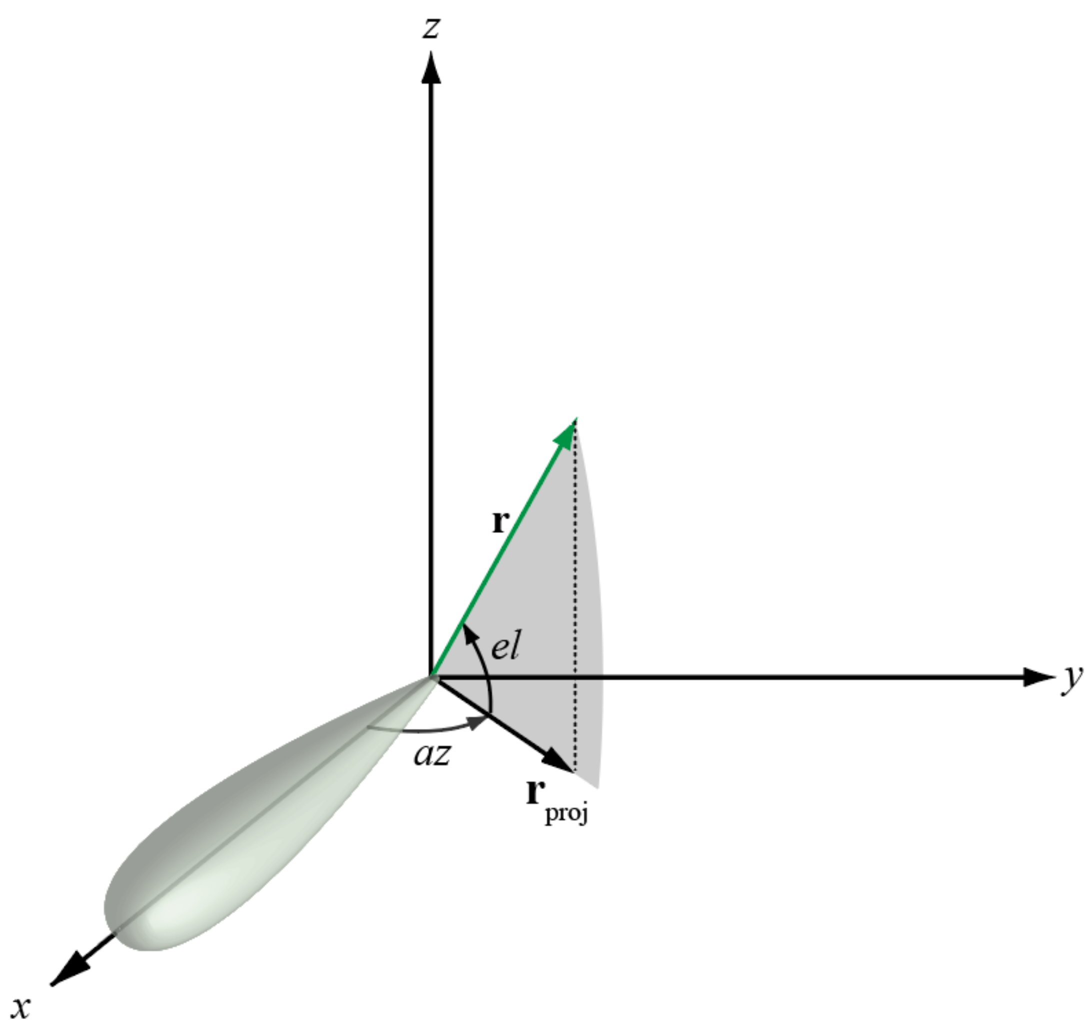
More About

Azimuth Angle, Elevation Angle

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



U/V Space

The *u* and *v* coordinates are the direction cosines of a vector with respect to the *y*-axis and *z*-axis, respectively.

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles on page 2-54 by:

$$u = \sin\theta\cos\phi$$
$$v = \sin\theta\sin\phi$$

In these expressions, φ and θ are the phi and theta angles, respectively.

To convert azimuth and elevation to *u* and *v* use the transformation

$$u = \cos el \sin az$$
$$v = \sin el$$

which is valid only in the range *abs(az)≤=90*.

The values of *u* and *v* satisfy the inequalities

$$-1 \leq u \leq 1$$
$$-1 \leq v \leq 1$$
$$u^2 + v^2 \leq 1$$

Conversely, the phi and theta angles can be written in terms of *u* and *v* using

$$\tan\phi = v/u$$
$$\sin\theta = \sqrt{u^2 + v^2}$$

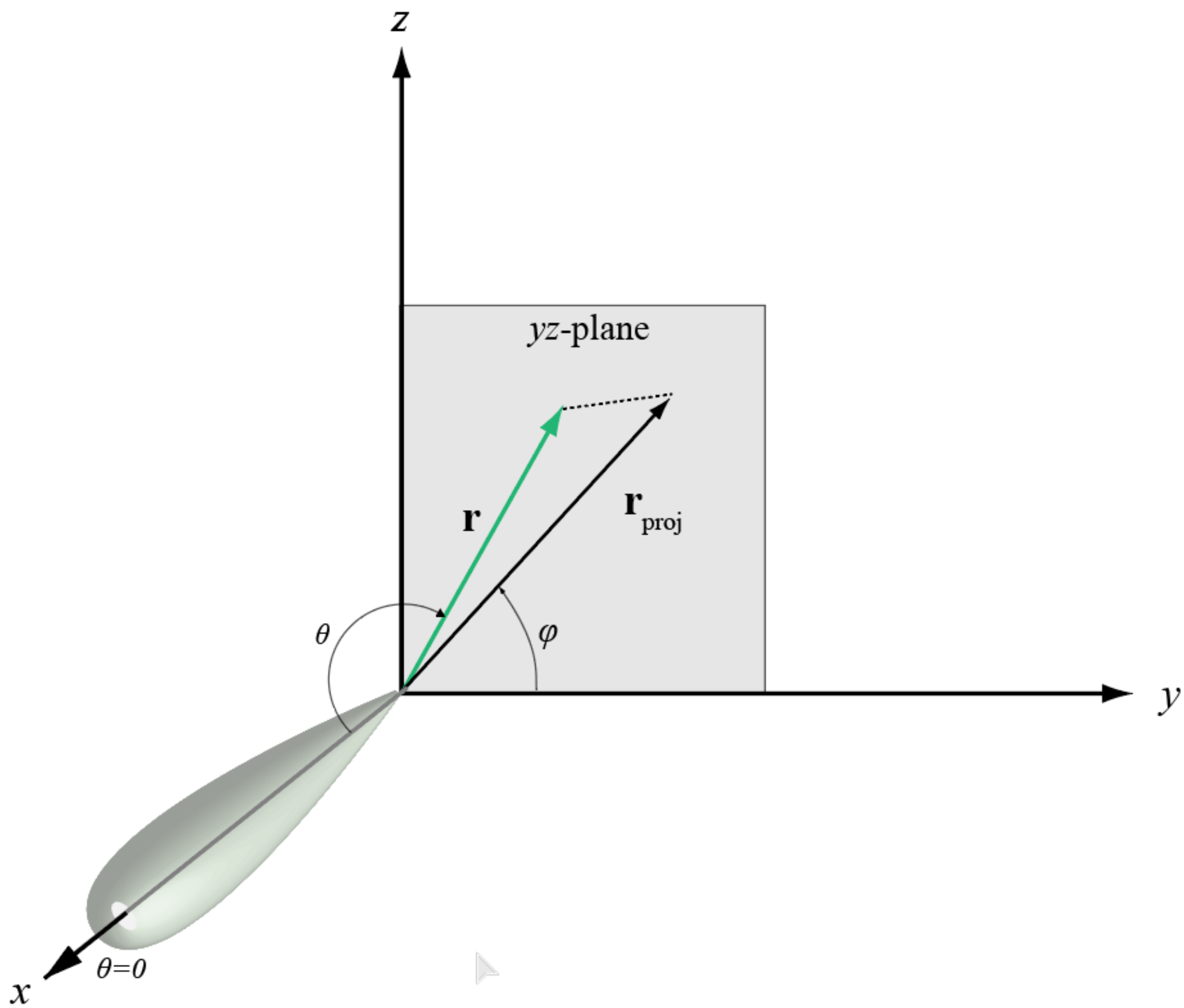
The azimuth and elevation angles can also be written in terms of *u* and *v*:

$$\sin el = v$$
$$\tan az = \frac{u}{\sqrt{1 - u^2 - v^2}}$$

Phi Angle, Theta Angle

The phi angle (φ) is the angle from the positive *y*-axis to the vector's orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi\sin\theta$

$\tan az = \cos\phi\tan\theta$

$\cos\theta = \cos el\cos az$

$\tan\phi = \tan el/\sin az$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
phased.CustomAntennaElement | azel2uv | uv2azel | uv2azelpat

Topics
"Spherical Coordinates"

azelcut2pat

Create 3-D response pattern from azimuth and elevation cuts

Syntax

```
pat = azelcut2pat(azcut,elcut)
```

Description

`pat = azelcut2pat(azcut,elcut)` creates a 3-D element response pattern, `pat`, from an azimuth cut, `azcut`, and an elevation cut, `elcut`. An azimuth cut consists of an antenna pattern over all azimuth angles at 0° elevation. An elevation cut consists of the antenna pattern over all elevation angles at 0° azimuth. You can specify cuts for different frequencies at the same time.

Examples

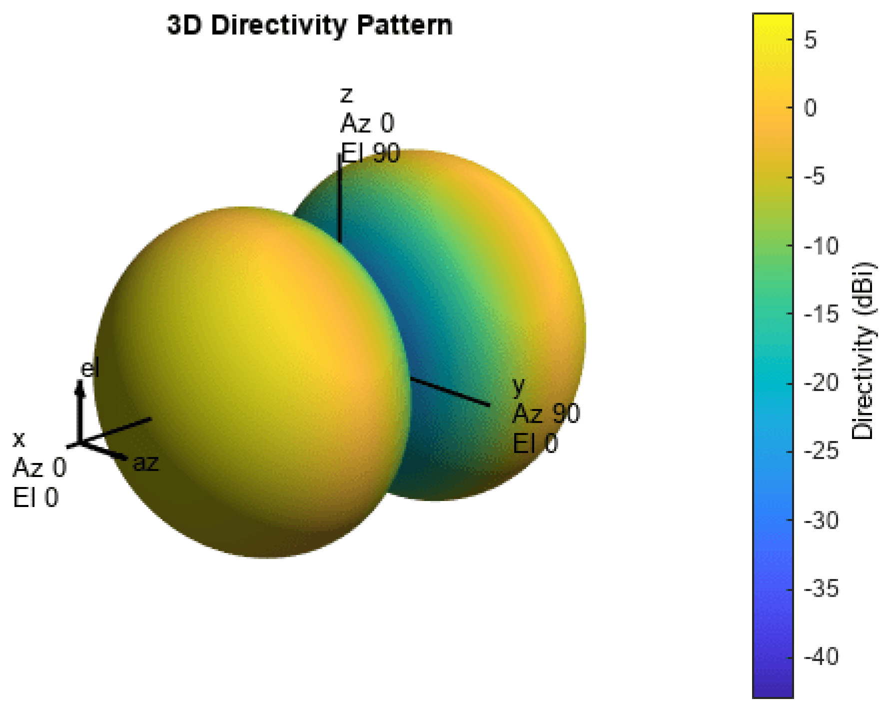
Create Custom Antenna Pattern from Azimuth and Elevations Cuts

Create a custom antenna pattern from azimuth and elevation cuts of a cosine-squared pattern.

```
az = -180:180;
azcut = mag2db(cosd(az).^2);
el = -90:90;
elcut = mag2db(cosd(el).^2);
pat = azelcut2pat(azcut,elcut);
antenna = phased.CustomAntennaElement('AzimuthAngles',az,...
    'ElevationAngles',el,'MagnitudePattern',pat,...
    'PhasePattern',zeros(size(pat)));
```

Display the antenna pattern for 200 MHz.

```
fs = 200.0e6;
pattern(antenna,fs);
```



Input Arguments

`azcut` — Azimuth pattern cut

`zeros(1,361)` (default) | real-valued 1-by-Q vector | real-valued L-by-Q matrix

Azimuth pattern cut, specified as a real-valued 1-by-Q vector or a real-valued L-by-Q matrix. Q is the number of azimuth angles, and L is the number of frequencies. Azimuth cuts are assumed to be made at 0° elevation. When `azcut` is a matrix, each column represents a different azimuth angle, and each row represents a different frequency. Units are in dB.

Data Types: `double`

`elcut` — Elevation pattern cut

`zeros(1,181)` (default) | real-valued 1-by-P vector | real-valued L-by-P matrix

Elevation pattern cut, specified as a real-valued 1-by-P vector or a real-valued L-by-P matrix. P is the number of elevation angles, and L is the number of frequencies. Elevation cuts are assumed to be made at 0° azimuth. When `elcut` is a matrix, each column represents a different elevation angle, and each row represents a different frequency. Units are in dB.

Data Types: `double`

Output Arguments

pat — 3-D antenna pattern
real-valued *P*-by-*Q* matrix | real-valued *P*-by-*Q*-by-*L* array

3-D array or antenna pattern, returned as a real-valued *P*-by-*Q* matrix or real-valued *P*-by-*Q*-by-*L* MATLAB array. Units are in dB.

Algorithms

The function returns a 3-D antenna pattern at the same azimuth and elevation angles used to define the `azcut` and `elcut` cuts. Because the cuts are specified in dB, the 3-D pattern is computed from the sum of the cut patterns.

pat(az,el) = pat(az) + pat(el)

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

azelaxes

Spherical basis vectors in 3-by-3 matrix form

Syntax

```
A = azelaxes(az,el)
```

Description

`A = azelaxes(az,el)` returns a 3-by-3 matrix containing the components of the basis($\hat{\mathbf{e}}_R$, $\hat{\mathbf{e}}_{az}$, $\hat{\mathbf{e}}_{el}$) at each point on the unit sphere specified by azimuth, `az`, and elevation, `el`. The columns of `A` contain the components of basis vectors in the order of radial, azimuthal and elevation directions.

Examples

Compute Spherical Basis Vectors

At the point located at 45° azimuth, 45° elevation, compute the 3-by-3 matrix containing the components of the spherical basis.

```
A = azelaxes(45,45)
```

```
A = 3×3

    0.5000   -0.7071   -0.5000
    0.5000    0.7071   -0.5000
    0.7071         0    0.7071
```

The first column of `A` contains the radial basis vector `[0.5000; 0.5000; 0.7071]`. The second and third columns are the azimuth and elevation basis vectors, respectively.

Input Arguments

az — Azimuth angle
scalar in range [–180,180]

Azimuth angle specified as a scalar in the closed range [–180,180]. Angle units are in degrees. To define the azimuth angle of a point on a sphere, construct a vector from the origin to the point. The azimuth angle is the angle in the *xy*-plane from the positive *x*-axis to the vector's orthogonal projection into the *xy*-plane. As examples, zero azimuth angle and zero elevation angle specify a point on the *x*-axis while an azimuth angle of 90° and an elevation angle of zero specify a point on the *y*-axis.

Example: 45

Data Types: `double`

el — Elevation angle
scalar in range [–90,90]

Elevation angle specified as a scalar in the closed range [–90,90]. Angle units are in degrees. To define the elevation of a point on the sphere, construct a vector from the origin to the point. The elevation angle is the angle from its orthogonal projection into the *xy*-plane to the vector itself. As examples, zero elevation angle defines the equator of the sphere and ±90° elevation define the north and south poles, respectively.

Example: `30`

Data Types: `double`

Output Arguments

A — Spherical basis vectors
3-by-3 matrix

Spherical basis vectors returned as a 3-by-3 matrix. The columns contain the unit vectors in the radial, azimuthal, and elevation directions, respectively. Symbolically we can write the matrix as

$$(\widehat{\mathbf{e}}_R, \widehat{\mathbf{e}}_{az}, \widehat{\mathbf{e}}_{el})$$

where each component represents a column vector.

More About

Spherical basis

Spherical basis vectors are a local set of basis vectors which point along the radial and angular directions at any point in space.

The spherical basis vectors $(\widehat{\mathbf{e}}_R, \widehat{\mathbf{e}}_{az}, \widehat{\mathbf{e}}_{el})$ at the point *(az,el)* can be expressed in terms of the Cartesian unit vectors by

$$\begin{aligned}
\widehat{\mathbf{e}}_{\mathbf{R}} &= \cos(el)\cos(az)\,\widehat{\mathbf{i}} + \cos(el)\sin(az)\,\widehat{\mathbf{j}} + \sin(el)\,\widehat{\mathbf{k}} \\
\widehat{\mathbf{e}}_{\mathbf{az}} &= -\sin(az)\,\widehat{\mathbf{i}} + \cos(az)\,\widehat{\mathbf{j}} \\
\widehat{\mathbf{e}}_{\mathbf{el}} &= -\sin(el)\cos(az)\,\widehat{\mathbf{i}} - \sin(el)\sin(az)\,\widehat{\mathbf{j}} + \cos(el)\,\widehat{\mathbf{k}}
\end{aligned}.$$

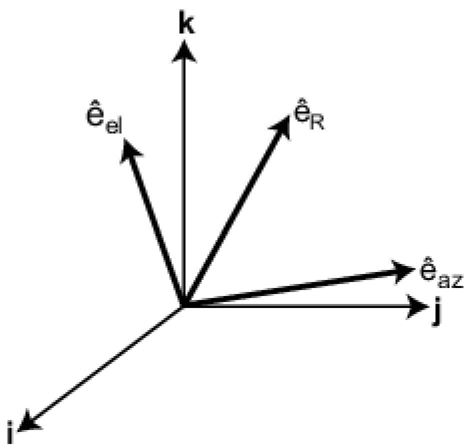
This set of basis vectors can be derived from the local Cartesian basis by two consecutive rotations: first by rotating the Cartesian vectors around the *y*-axis by the negative elevation angle, *-el*, followed by a rotation around the *z*-axis by the azimuth angle, *az*. Symbolically, we can write

$$\hat{\mathbf{e}}_{\mathbf{R}} = R_z(az)R_y(-el)\begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$\hat{\mathbf{e}}_{\mathbf{az}} = R_z(az)R_y(-el)\begin{bmatrix}0\\1\\0\end{bmatrix}$$

$$\hat{\mathbf{e}}_{\mathbf{el}} = R_z(az)R_y(-el)\begin{bmatrix}0\\0\\1\end{bmatrix}$$

The following figure shows the relationship between the spherical basis and the local Cartesian unit vectors.



Algorithms

MATLAB computes the matrix A from the equations

```
A = [cosd(el)*cosd(az), -sind(az), -sind(el)*cosd(az); ...
        cosd(el)*sind(az),  cosd(az), -sind(el)*sind(az); ...
        sind(el),           0,         cosd(el)];
```

Version History

Introduced in R2013a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`cart2sphvec` | `sph2cartvec`

beamwidth2gain

Compute antenna gain from azimuth and elevation beamwidths

Syntax

```
g = beamwidth2gain(hpbw)
g = beamwidth2gain(hpbw,at)
```

Description

`g = beamwidth2gain(hpbw)` returns the gain of an antenna given its half-power beamwidth, `hpbw`.

`g = beamwidth2gain(hpbw,at)` also specifies the type of antenna aperture.

Examples

Antenna Gain for Uniformly Illuminated Rectangular Aperture

Compute the antenna gain for a uniformly illuminated rectangular aperture. Specify an azimuth beamwidth of `1.4` degrees and an elevation beamwidth of `5` degrees.

```
BWaz = 1.4;
BWel = 5;

g = beamwidth2gain([BWaz;BWel],'UniformRectangular')
```

```
g = 36.6522
```

Input Arguments

`hpbw` — Antenna half-power beamwidth
row vector | two-row matrix

Antenna half-power beamwidth in degrees, specified as a row vector or a two-row matrix.

- If `hpbw` is a row vector, then `beamwidth2gain` assumes a symmetric aperture and each element of `hpbw` specifies the same beamwidth for both azimuth and elevation dimensions.
- If `hpbw` is a two-row matrix, then its first row contains azimuth beamwidth values and its second row contains elevation beamwidth values.

Data Types: `double` | `single`

`at` — Antenna aperture type
`'IdealRectangular'` (default) | `'IdealElliptical'` | `'IdealGaussian'` | `'UniformRectangular'` | `'CosineRectangular'` | `'UniformCircular'` | `'ParabolicCircular'` | `'PracticalGeneral'`

Antenna aperture type, specified as one of these:

- 'IdealRectangular' — Rectangular beam with no sidelobes
- 'IdealElliptical' — Elliptical beam with no sidelobes
- 'IdealGaussian' — Gaussian beam with no sidelobes
- 'UniformRectangular' — Uniformly illuminated rectangular aperture
- 'CosineRectangular' — Cosine illuminated rectangular aperture
- 'UniformCircular' — Uniformly illuminated circular aperture. In this case, hpbw must be either a row vector or a two-row matrix with identical rows because the beamwidth is the same in the azimuth and elevation dimensions.
- 'ParabolicCircular' — Circular aperture parabolic-on-a-12 dB pedestal distribution. In this case, hpbw must be either a row vector or a two-row matrix with identical rows because the beamwidth is the same in the azimuth and elevation dimensions.
- 'PracticalGeneral' — General-use practical antenna with sidelobes and null fill. For more details, see [1].

Data Types: char | string

Output Arguments

g — Antenna gain
vector

Antenna gain in dBi units, returned as a vector. The number of elements of g equals the number of columns of hpbw.

Version History

Introduced in R2021a

References

[1] Stutzman, Warren L., and Gary A. Thiele. *Antenna Theory and Design*. 3rd ed. Hoboken, NJ: Wiley, 2013.

[2] Stutzman, Warren L. “Estimating Directivity and Gain of Antennas.” *IEEE Antennas and Propagation Magazine* 40, no. 4 (August 1998): 7–11. https://doi.org/10.1109/74.730532.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

aperture2gain | gain2aperture

beat2range

Convert beat frequency to range

Syntax

```
r = beat2range(fb,slope)
r = beat2range(fb,slope,c)
```

Description

`r = beat2range(fb,slope)` converts the beat frequency on page 2-67 of a dechirped linear FMCW signal to its corresponding range. `slope` is the slope of the FMCW sweep.

`r = beat2range(fb,slope,c)` specifies the signal propagation speed.

Examples

Range of Target in FMCW Radar System

Assume that an FMCW waveform sweeps a band of 3 MHz in 2 ms. The dechirped target return has a beat frequency of 1 kHz. Compute the target range.

```
slope = 30e6/(2e-3);
fb = 1e3;
r = beat2range(fb,slope)
```

```
r = 9.9931
```

Input Arguments

`fb` — Beat frequency of dechirped signal
M-by-1 vector | M-by-2 matrix

Beat frequency of dechirped signal, specified as an M-by-1 vector or M-by-2 matrix in hertz. If the FMCW signal performs an upsweep or downsweep, `fb` is a vector of beat frequencies.

If the FMCW signal has a triangular sweep, `fb` is an M-by-2 matrix in which each row represents a pair of beat frequencies. Each row has the form `[UpSweepBeatFrequency,DownSweepBeatFrequency]`.

Data Types: `single` | `double`

`slope` — Sweep slope
nonzero scalar

Slope of FMCW sweep, specified as a nonzero scalar in hertz per second. If the FMCW signal has a triangular sweep, `slope` is the sweep slope of the up-sweep half. In this case, `slope` must be positive and the down-sweep half is assumed to have a slope of `-slope`.

Data Types: `single` | `double`

c — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar in meters per second.

Data Types: `single` | `double`

Output Arguments

r — Range
M-by-1 column vector

Range, returned as an M-by-1 column vector in meters. Each row of `r` is the range corresponding to the beat frequency in a row of `fb`.

Data Types: `single` | `double`

More About

Beat Frequency

For an up-sweep or down-sweep FMCW signal, the beat frequency is $F_t - F_r$. In this expression, F_t is the transmitted signal's carrier frequency, and F_r is the received signal's carrier frequency.

For an FMCW signal with triangular sweep, the upsweep and downsweep have separate beat frequencies.

Algorithms

If `fb` is a vector, the function computes `c*fb/(2*slope)`.

If `fb` is an M-by-2 matrix with a row `[UpSweepBeatFrequency,DownSweepBeatFrequency]`, the corresponding row in `r` is `c*((UpSweepBeatFrequency - DownSweepBeatFrequency)/2)/(2*slope)`.

This function supports single and double precision for input data and arguments.

Version History

Introduced in R2012b

References

[1] Pace, Phillip. *Detecting and Classifying Low Probability of Intercept Radar*. Artech House, Boston, 2009.

[2] Skolnik, M.I. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

- Usage notes and limitations:
- Does not support variable-size inputs.
- This function supports single and double precision for input data and arguments.

See Also
`dechirp` | `range2beat` | `rdcoupling` | `phased.FMCWWaveform`

Topics
"Automotive Adaptive Cruise Control Using FMCW Technology" (Radar Toolbox)

jsdmbbweights

Create MU-MIMO baseband precoding weights

Syntax

```
wpbb = jsdmbbweights(chaneff,ns,wprf)
wpbb = jsdmbbweights( ___ ,'RegularizationFactor',alpha)
[wpbb,wc] = jsdmbbweights( ___ )
```

Description

`wpbb = jsdmbbweights(chaneff,ns,wprf)` returns baseband precoding weights, `wpbb`, for the effective channel, `chaneff`, to achieve orthogonality among users by Joint Spatial Division Multiplexing (JSDM). RF precoding weights, `wprf`, divide users into groups by their location information. `ns` is the number of data streams or users.

`wpbb = jsdmbbweights( ___ ,'RegularizationFactor',alpha)` specifies the regularization factor, `alpha`.

`[wpbb,wc] = jsdmbbweights( ___ )` also returns combining weights `wc` for each user.

Examples

Find JSDM Baseband Precoding Weights

Find baseband precoding weights for a JSDM beamformer based on a base station with a 64-element linear antenna array. Assume there are two user groups. The first group has two users and the second group has three users. Array elements are 1/2 wavelength apart.

Set the number of array elements to 64.

```
Nt = 64;
```

Find the array steering weights in the direction of the first user group. The user group is centered at $+10^{\circ}$azimuth.

```
v1 = steervec((0:Nt-1)*0.5,[9 11]);
rc1 = v1*v1';
```

Find the array steering weights in the direction of the second user group. The user group is centered -5°azimuth.

```
v2 = steervec((0:Nt-1)*0.5,[-4 -5 -6]);
rc2 = v2*v2';
```

Then, compute the RF weights from the channel covariance matrix.

```
ns = [2 3];
hcov = {rc1,rc2};
wprf = jsdmrfweights(hcov,ns);
```

Construct the instantaneous effective channel.

```
heff1 = wprf{1}*v1*eye(ns(1))*randn(ns(1),ns(1),'like',1i);
heff2 = wprf{2}*v2*eye(ns(2))*randn(ns(2),ns(2),'like',1i);
```

Compute baseband weights using per-group-processing.

```
[wpbb,wc] = jsdmbbweights({heff1,heff2},ns,wprf);
```

Verify orthogonality.

```
hortho1 = wpbb{1}*heff1*wc{1}
```

```
hortho1 = 2×2 complex

   9.5489 - 0.0000i   0.0000 + 0.0000i
   0.0000 - 0.0000i   9.5489 + 0.0000i
```

```
hortho2 = wpbb{2}*heff2*wc{2}
```

```
hortho2 = 3×3 complex

   2.8633 + 0.0000i   0.0000 - 0.0000i   0.0000 + 0.0000i
  -0.0000 + 0.0000i   2.8633 + 0.0000i  -0.0000 - 0.0000i
   0.0000 - 0.0000i  -0.0000 - 0.0000i   2.8633 - 0.0000i
```

Input Arguments

`chaneff` — Effective downlink channels

complex-valued *Nsc*-by-*TotalNrtf*-by-*TotalNs* array | 1-by-*G* cell array

Effective downlink channels, specified as a complex-valued *Nsc*-by-*TotalNrtf*-by-*TotalNs* array or 1-by-*G* cell array. The effective downlink channels map the streams in each group to the *G* user groups.

- If `chaneff` is a 3-dimensional *Nsc*-by-*TotalNtrf*-by-*TotalNs* array, *TotalNs* is the total number of data streams, *TotalNtrf* is the sum of RF chains for all user groups, and *Nsc* is the number of subcarriers. This configuration corresponds to the joint group processing (JGP) scheme of JSDM.
- If `chaneff` is a 1-by-*G* cell array, then the g^{th} element in the cell array contains the individual effective channel from the allocated RF chain at the base station to the corresponding group of users as either:

  - an *Nsc*-by-*Ntrf(g)*-by-*NS(g)* array when `ns` is a vector, or
  - an *Nsc*-by-*Ntrf(g)*-by-sum*(NS{g})* array when `ns` is a cell array.

  This configuration corresponds to the per group processing (PGP) scheme of JSDM.

Data Types: `double` | `cell`
Complex Number Support: Yes

`ns` — Number of data streams

G-element row vector (default) | 1-by-*G* cell array

Number of data streams, specified as a *G*-element row vector or a 1-by-*G* cell array.

- If `ns` is a vector, each element specifies the total number of data streams in the corresponding user group.
- If `ns` is a cell array, each element in the cell array is a K_g-element row vector where K_g is the number of users in the g^{th} group. The elements in the K_g-element row vector specifies the number of receiving data streams for each user in that group.

Data Types: `double` | `cell`

`wprf` — RF precoding weights
1-by-*G* cell array

RF precoding weights, specified as an 1-by-*G* cell array. The g^{th} element in `wprf` is an *Ntrf(g)*-by-*Nt* matrix that maps *Ntrf(g)* RF chain to *Nt* antennas in the base station. *Ntrf(g)* is the number of RF chains dedicated to the g^{th} group.

Data Types: `cell` | `double`
Complex Number Support: Yes

`alpha` — Regularization factor
0 (default) | nonnegative scalar

Regularization factor, specified as a nonnegative scalar. Units are dimensionless.

Data Types: `double`

Output Arguments

`wpbb` — Baseband precoding weights
Nsc-by-*TotalNS*-by-*TotalNtrf* array | 1-by-*G* cell array

Baseband precoding weights, returned as a complex-valued *Nsc*-by-*TotalNS*-by-*TotalNtrf* array or 1-by-*G* cell array.

- If `chaneff` is not a cell array, `wpbb` is an *Nsc*-by-*TotalNS*-by-*TotalNtrf* array.
- If `chaneff` is a cell array, then `wpbb` is a 1-by-*G* cell array whose g^{th} element is either an *Nsc*-by-*Ns(g)*-by-*Ntrf(g)* array when `ns` is a vector or an *Nsc*-by-*sum(NS{g})*-by-*Ntrf(g)* array when `ns` is a cell array.

.

`wc` — Recombining weights
1-by-*G* cell array

Recombining weights for each user, returned as a 1-by-*G* cell array. The structure of `wc` depends on the `ns` input argument.

- If `ns` is a vector, each cell in `wc` is an *Nsc*-by-*NS(g)*-by-*NS(g)* array.
- If `ns` is a cell array, then `wc`*(g)* is a 1-by-K_g cell array where K_g is the number of users in the g^{th} user group. Within the K_g-element cell array, each entry is an *Nsc*-by-*sum(Ns{g})*-by-*Ns{g}(k)* array where *Ns{g}(k)* is the number of data streams for the k^{th} user in the g^{th} user group.

Version History
Introduced in R2023a

References

[1] Adhikary A., J. Nam, J-Y Ahn, and G. Caire. "Joint Spatial Division and Multiplexing - The Large-Scale Array Regime." *IEEE Transactions on Information Theory*, Vol. 59, No. 10, October 2013, pp. 6441-6463.

See Also

`jsdmrfweights` | `blkdiagbfweights` | `omphybweights`

Topics

"Massive MIMO Hybrid Beamforming"

jsdmrfweights

Create RF precoding weights

Syntax

```
wprf = jsdmrfweights(hcov,ns)
wprf = jsdmrfweights(hcov,ns,ntrf)
```

Description

`wprf = jsdmrfweights(hcov,ns)` returns the RF precoding weights, `wprf`, for the channel covariance matrix, `hcov`, to achieve orthogonality among users by Joint Spatial Division Multiplexing (JSDM). JSDM uses RF precoding weights to divide users into groups by their location information. The RF precoding weights, `wprf`, beamform the transmitted signal into each group.

`wprf = jsdmrfweights(hcov,ns,ntrf)` also specifies the desired number of RF chains `ntrf`.

Examples

Find JSDM RF Precoding Weights

Find RF precoding weights for a JSDM beamformer based on a base station with a 64-element linear antenna array. Assume there are two user groups. The first group has two users and the second group has three users. Array elements are 1/2 wavelength apart.

Set the number of array elements to 64.

```
Nt = 64;
```

Find the array steering weights in the direction of the first user group. The user group is centered at $+10^{\circ}$ azimuth.

```
v1 = steervec((0:Nt-1)*0.5,[9 11]);
rc1 = v1*v1';
```

Find the array steering weights in the direction of the second user group. The user group is centered -5° azimuth.

```
v2 = steervec((0:Nt-1)*0.5,[-4 -5 -6]);
rc2 = v2*v2';
```

Then, compute the RF weights from the channel covariance matrix.

```
hcov = {rc1,rc2};
wprf = jsdmrfweights(hcov,[2 3]);
```

Verify the orthogonality between the two user groups.

```
snr1 = pow2db(real(trace(wprf{1}*rc1*wprf{1}'))/real(trace(wprf{1}*rc2*wprf{1}')))
```

```
snr1 = 26.1255
```

```
snr2 = pow2db(real(trace(wprf{2}*rc2*wprf{2}'))/real(trace(wprf{2}*rc1*wprf{2}')))
```

```
snr2 = 26.1632
```

Input Arguments

hcov — Channel covariance matrix
1-by-*G* cell array

Channel covariance matrix, specified as 1-by-*G* cell array where *G* is the number of groups. Each cell in `hcov` an N_t-by-N_t matrix representing the covariance of the channel between the base station and the corresponding group of the users. N_t is the number of elements in the base station array. Assume that `hcov` is the same for all users in the corresponding group across all subcarriers.

`hcov` for the g^{th} user group is defined as the expected value of *H*H'*, where *H* is an N_t-by-$N_r(g)$ matrix representing the instantaneous channel for that user group. $N_r(g)$ is the number of receiving elements in theg^{th} group.

Data Types: `double`
Complex Number Support: Yes

ns — Number of data streams
G-element row vector | 1-by-*G* cell array

Number of data streams, specified as a *G*-element row vector or a 1-by-*G* cell array.

- If `ns` is a vector, each element specifies the total number of data streams in the corresponding user group.
- If `ns` is a cell array, each element in the cell array is a K_g-element row vector where K_g is the number of users in the g^{th} group. The elements in K_g-element row vector specify the number of receiving data streams for each user in that group.

This function assumes that the number of receiving elements in the g^{th} user group, $N_r(g)$, matches the total number of data streams for that group.

Data Types: `double`

ntrf — Desired number of RF chains
number of data streams per group (default) | 1-by-*G* integer vector

Desired number of RF chains for each group, specified as a 1-by-*G* integer vector. Note that if `nrtf` is specified, then the dimension of `wprf`*(g)* becomes `nrtf`*(g)*-by-N_t.

Data Types: `double`

Output Arguments

wprf — RF precoding weights
1-by-*G* cell array

RF precoding weights for the channel covariance matrix, returned as a 1-by-*G* cell array.

- When `ns` is a vector, each cell in `wprf` is an `ns`*(g)*-by-N_t matrix that maps the `ns`*(g)* RF chain to *Nt* antennas in the base station.

- When `ns` is a cell array, each cell in `wprf` is a *sum(NS{g})-by-Nt* matrix that maps the total number of RF chains in the g^{th} group to *Nt* antennas in the base station.

Version History

Introduced in R2023a

References

[1] Adhikary A., J. Nam, J-Y Ahn, and G. Caire. "Joint Spatial Division and Multiplexing - The Large-Scale Array Regime." *IEEE Transactions on Information Theory*, Vol. 59, No. 10, October 2013, pp. 6441-6463.

See Also

`jsdmbbweights` | `blkdiagbfweights` | `omphybweights`

Topics

"Massive MIMO Hybrid Beamforming"

blkdiagbfweights

MIMO channel block diagonalized weights

Syntax

```
[wp,wc] = blkdiagbfweights(chanmat,ns)
[wp,wc] = blkdiagbfweights(chanmat,ns,pt)
```

Description

`[wp,wc] = blkdiagbfweights(chanmat,ns)` returns precoding weights, `wp`, and combining weights, `wc`, derived from the channel response matrices contained in a MATLAB cell array `chanmat`.

- You can specify multiple user channels by putting each channel in a `chanmat` cell. `chanmat{k}` represents the k^{th}channel from the transmitter to the user.
- - For a single frequency, specify the channel cell as a matrix.
  - For multiple frequencies, specify the channel cell as a three-dimensional array where the rows represent different subcarriers.
- Specify multiple subchannels per channel using the `ns` argument. Subchannels represent different data streams. `ns` specifies the number of subchannels for each user channel. Multiply the data streams by the precoding weights, `wp`.

The precoding and combining weights diagonalize the channel into independent subchannels so that for the k^{th}user, the matrix `wp*chanmat{k}*wc{k}` is diagonal for each subcarrier.

`[wp,wc] = blkdiagbfweights(chanmat,ns,pt)` also specifies the total transmitted power, `pt`, per subcarrier.

Examples

Spatial Multiplexing with Block Diagonal Weights

Start with a base station consisting of a uniform linear array (ULA) with 16 antennas, and two users having receiver ULA arrays with 8 and 4 antennas, respectively. Show that using block diagonalization-based precoding and combining weights achieves spatial multiplexing, where the received signal at each user can be decoded without interference from the other user. Specify two data streams for each user.

Specify the transmitter location in `txpos` and two user receiver locations in `rxpos1` and `rxpos2`. Array elements are spaced one-half wavelength apart.

```
txpos = (0:15)*0.5;
rxpos1 = (0:7)*0.5;
rxpos2 = (0:3)*0.5;
```

Create the channel matrix cell array using `scatteringchanmtx` and then compute the beamforming weights `wp` and `wc`. Each channel corresponds to a user. Assume that the channels have 10 scatterers. Each channel has two subchannels specified by the vector `ns`.

```
chanmat = {scatteringchanmtx(txpos,rxpos1,10), ...
    scatteringchanmtx(txpos,rxpos2,10)};
ns = [2 2];
[wp,wc] = blkdiagbfweights(chanmat,ns);
```

The weights diagonalize the channel matrices for each user.

For channel 1:

```
disp(wp*chanmat{1}*wc{1})
```

```
   8.2269 - 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i   6.1371 - 0.0000i
   0.0000 - 0.0000i  -0.0000 + 0.0000i
   0.0000 - 0.0000i   0.0000 + 0.0000i
```

For channel 2:

```
disp(wp*chanmat{2}*wc{2})
```

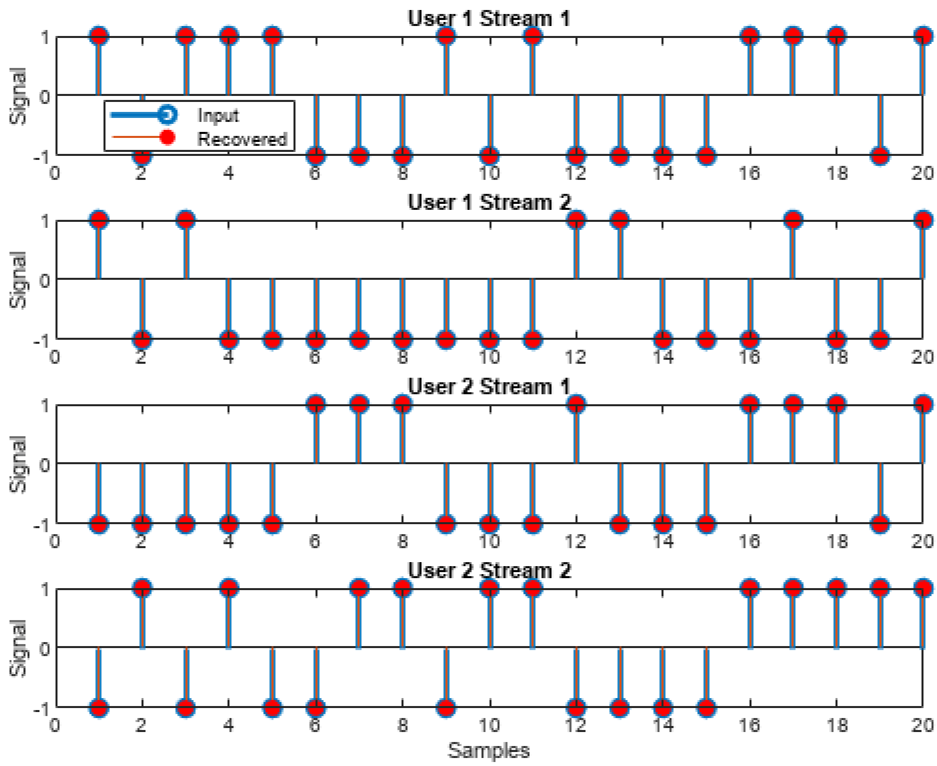
```
  -0.0000 + 0.0000i  -0.0000 + 0.0000i
  -0.0000 + 0.0000i  -0.0000 + 0.0000i
   8.7543 - 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i   4.4372 + 0.0000i
```

First create four subchannels to carry the data streams: two subchannels per channel. Each data stream contains 20 samples of ±1. Precode the input streams and combine the streams to produce the recovered signals.

```
x = 2*round(rand([20,4])) - 1;
xp = x*wp;
y1 = xp*chanmat{1} + 0.1*randn(20,8);
y2 = xp*chanmat{2} + 0.1*randn(20,4);
y = [y1*wc{1},y2*wc{2}];
```

Overlay stem plots of the input and recovered signals to show that the received user signals are the same as the transmitted signals.

```
for m = 1:4
    subplot(4,1,m)
    s = stem([x(:,m) 2*((real(y(:,m)) > 0) - 0.5)]);
    s(1).LineWidth = 2;
    s(2).MarkerEdgeColor = 'none';
    s(2).MarkerFaceColor = 'r';
    ylabel('Signal')
    title(sprintf('User %d Stream %d',ceil(m/2),rem(m-1,2) + 1))
    if m==1
        legend('Input','Recovered','Location','best')
    end
end
xlabel('Samples')
```



Spatial Multiplexing with Specified Power

Start with a base station consisting of a uniform linear array (ULA) with 16 antennas, and two users having receiver ULA arrays with 8 and 5 antennas, respectively. Show how to use three-dimensional arrays of channel matrices to handle two subcarriers. Then, the channel matrix for the first user takes the form 2-by-16-by-8 and the channel matrix for the second user takes the form 2-by-16-by-5. Also assume that there are two data streams for each user.

Specify the transmitter location in `txpos` and two user receiver locations in `rxpos1` and `rxpos2`. Array elements are spaced one-half wavelength apart.

```
nr1 = 8;
nr2 = 5;
txpos = (0:15)*0.5;
rxpos1 = (0:(nr1-1))*0.5;
rxpos2 = (0:(nr2-1))*0.5;
```

Create the channel matrices using `scatteringchanmtx` and put them in a cell array. To create a second subchannel for each receiver, duplicate each channel matrix. Assume 10 point scatterers in computing the channel matrix.

```
smtmp1 = scatteringchanmtx(txpos,rxpos1,10);
smtmp2 = scatteringchanmtx(txpos,rxpos2,10);
sm1 = zeros(2,16,8);
```

```
sm2 = zeros(2,16,5);
sm1(1,:,:) = smtmp1;
sm1(2,:,:) = smtmp1;
sm2(1,:,:) = smtmp2;
sm2(2,:,:) = smtmp2;
chanmat = {sm1,sm2};
```

Specify that there are two data streams for each user.

```
ns = [2 2];
```

Specify the transmitted powers for each subcarrier.

```
pt = [1.0 1.5];
```

Compute the beamforming weights.

```
[wp,wc] = blkdiagbfweights(chanmat,ns,pt);
```

Show that the channels are diagonalized for the first subcarrier.

```
ksubcr = 1;
wpx = squeeze(wp(ksubcr,:,:));
chanmat1 = squeeze(chanmat{1}(ksubcr,:,:));
chanmat2 = squeeze(chanmat{2}(ksubcr,:,:));
wc1 = squeeze(wc{1}(ksubcr,:,:));
wc2 = squeeze(wc{2}(ksubcr,:,:));
wpx*chanmat1*wc1
```

```
ans = 4×2 complex

   8.2104 + 0.0000i  -0.0000 - 0.0000i
   0.0000 - 0.0000i   5.9732 - 0.0000i
   0.0000 - 0.0000i  -0.0000 - 0.0000i
   0.0000 - 0.0000i   0.0000 - 0.0000i
```

```
wpx*chanmat2*wc2
```

```
ans = 4×2 complex

  -0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.0000 + 0.0000i   0.0000 + 0.0000i
   8.8122 + 0.0000i  -0.0000 + 0.0000i
   0.0000 + 0.0000i   4.8186 - 0.0000i
```

Propagate the signals to each user and then decode. Generate four streams of random data containing -1's and +1's and having two columns for each user. Each stream is a subchannel.

```
x = 2*(round(rand([20 4]))) - 1;
```

Precode the data streams.

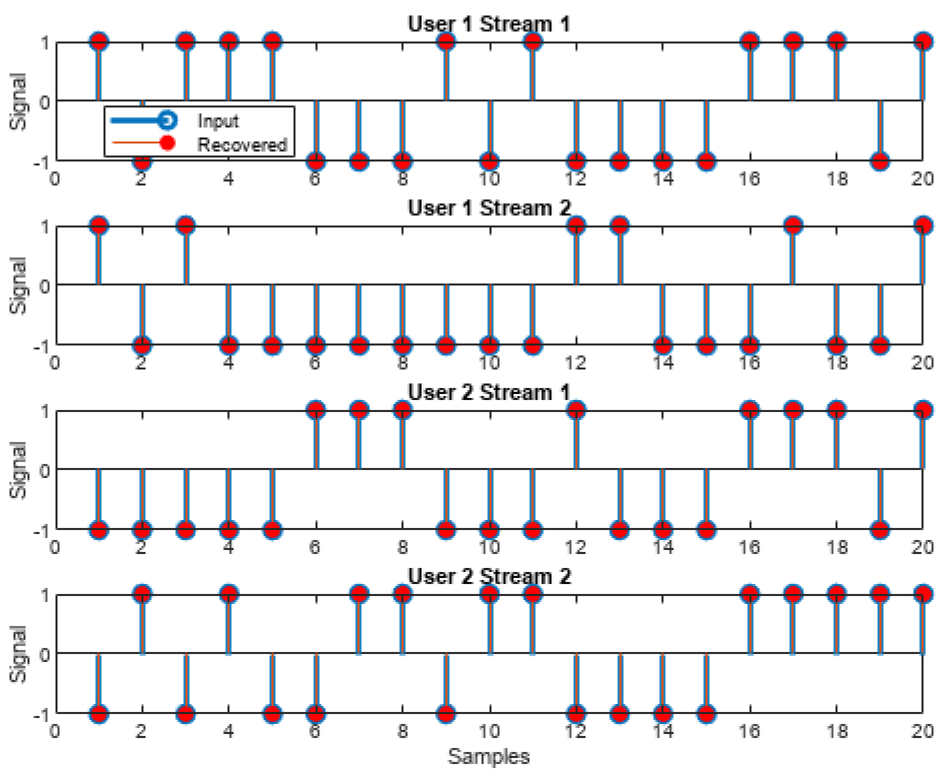
```
xp = x*wpx;
y1 = xp*chanmat1 + 0.1*randn(20,8);
y2 = xp*chanmat2 + 0.1*randn(20,5);
```

Decode the data streams.

```
y = [y1*wc1,y2*wc2];
```

Overlay stem plots of the input and recovered signals to show that the received user signals are the same as the transmitted signals.

```
for m = 1:4
    subplot(4,1,m)
    s = stem([x(:,m) 2*((real(y(:,m)) > 0) - 0.5)]);
    s(1).LineWidth = 2;
    s(2).MarkerEdgeColor = 'none';
    s(2).MarkerFaceColor = 'r';
    ylabel('Signal')
    title(sprintf('User %d Stream %d',ceil(m/2),rem(m-1,2) + 1))
    if m==1
        legend('Input','Recovered','Location','best')
    end
end
xlabel('Samples')
```



Input Arguments

`chanmat` — Channel response matrices

N_u-element cell array

Channel response matrices, specified as an N_u-element cell array. N_u is the number of receive arrays. Each cell corresponds to a different channel and contains a channel response matrix or a three dimensional MATLAB array. The cell array must contain either all matrices or all arrays. For matrices, the number of rows for all matrices must be the same. For three-dimensional arrays, the number of rows and columns must be the same.

- If the k^{th} cell is a matrix, the matrix has the size N_t-by-$N_r(k)$. N_t is the number of elements in the transmitting array and $N_r(k)$ is the number of elements in the k^{th} receiving array.
- If the k^{th} cell is an array, the array has the size L-by-N_t-by-$N_r(k)$. L is the number of subcarriers. N_t is the number of elements in the transmit array and $N_r(k)$ is the number of elements in the k^{th} receive array.

Data Types: `double`
Complex Number Support: Yes

`ns` — Number of data streams per receive array
N_u-element row vector of positive integers

Number of data streams per receive array, specified as an N_u-element row vector of positive integers. N_u is the number of receive arrays.

Data Types: `double`

`pt` — Total transmitted power per subcarrier
`1` (default) | positive scalar | L-element vector of positive values

Total transmitted power per subcarrier, specified as a positive scalar or an L-element vector of positive values. L is the number of subcarriers. If `pt` is a scalar, all subcarriers have the same transmitted power. If `pt` is a vector, each vector element specifies the transmitted power for the corresponding subcarrier. Power is in linear units.

Data Types: `double`

Output Arguments

`wp` — Precoding weights
complex-valued N_{st}-by-N_t matrix | complex-valued L-by-N_{st}-by-N_t MATLAB array

Precoding weights, returned as a complex-valued N_{st}-by-N_t matrix or a complex-valued L-by-N_{st}-by-N_t MATLAB array.

- If `chanmat` contains matrices, `wp` is a complex-valued N_{st}-by-N_t matrix where N_{st} is the total number of data channels (`sum(ns)`).
- If `chanmat` contains three-dimensional MATLAB arrays, `wp` is a complex-valued L-by-N_{st}-by-N_t MATLAB array where N_{st} is the total number of data channels (`sum(ns)`).

Units are dimensionless.

Data Types: `double`

`wc` — Combining weights
N_u-element cell array

Combining weights, returned as an N_u-element cell array. Units are dimensionless.

- If `chanmat` contains matrices, the k^{th} cell in `wc` contains a complex valued $N_r(k)$-by-$N_s(k)$ matrix. $N_s(k)$ is the value of the argument `ns` for the k^{th} receive array.

- If `chanmat` contains three-dimensional MATLAB arrays, the k^{th} cell of `wc` contains a complex-valued L-by-$N_r(k)$-by-$N_s(k)$ MATLAB array. $N_s(k)$ is the value of the k^{th} entry of the `ns` vector.

Data Types: `double`

Version History

Introduced in R2020a

References

[1] Heath, Robert W., et al. “An Overview of Signal Processing Techniques for Millimeter Wave MIMO Systems.” *IEEE Journal of Selected Topics in Signal Processing*, vol. 10, no. 3, Apr. 2016, pp. 436–53. DOI.org (Crossref), doi:10.1109/JSTSP.2016.2523924. Bibliography

[2] Tse, D. and P. Viswanath, *Fundamentals of Wireless Communications*, Cambridge: Cambridge University Press, 2005.

[3] Paulraj, A. *Introduction to Space-Time Wireless Communications*, Cambridge: Cambridge University Press, 2003.

[4] Spencer, Q.H., et al. "Zero-Forcing Methods for Downlink Spatial Multiplexing in Multiuser MIMO Channels." *IEEE Transactions on Signal Processing*, Vol. 52, No. 2, February 2004, pp. 461-471. DOI.org (Crossref), doi:10.1109/TSP.2003.821107.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.

See Also

Functions

`diagbfweights` | `scatteringchanmtx` | `waterfill`

Objects

`phased.ScatteringMIMOChannel`

broadside2az

Convert broadside angle to azimuth angle

Syntax

```
az = broadside2az(bsang)
az = broadside2az(bsang,el)
```

Description

`az = broadside2az(bsang)` returns the azimuth angle on page 2-84, `az`, corresponding to the broadside angle, `bsang`, for zero elevation angle. Angles are defined with respect to the local coordinate system.

`az = broadside2az(bsang,el)` also specifies the elevation angle, `el`.

Examples

Convert Broadside Angle to Azimuth Angle at Zero Elevation

Return the azimuth angle corresponding to a broadside angle of 45° at 0° elevation.

```
az = broadside2az(45.0)
```

```
az = 45.0000
```

Convert Broadside Angle to Azimuth Angle

Return the azimuth angle corresponding to a broadside angle of 45° and an elevation angle of 20°.

```
az = broadside2az(45,20)
```

```
az = 48.8063
```

Convert Multiple Broadside Angles to Azimuth Angles

Return azimuth angles for 10 pairs of broadside angle and elevation angle.

```
BSang = (45:5:90)';
el = (45:-5:0)';
az = broadside2az(BSang,el);
```

Input Arguments

bsang — Broadside angle
scalar | vector of real values

Broadside angle, specified as a scalar or vector of real values. Units are in degrees. This argument supports single and double precision.

Example: `[10;-22;-80]`

el — Elevation angle
`0` (default) | scalar | vector of real values

Elevation angle, specified as a scalar or vector of real values. The length of `el` must match the length of `bsang`. Elevation angles lie in the range from –90° to 90°. Units are in degrees. This argument supports single and double precision.

Example: `[5;2;-1]`

Output Arguments

az — Azimuth angle
scalar | vector of real values

Azimuth angle, returned as a scalar or vector of real values. The length of `az`equals the length of `bsang`. Azimuth angles lie in the range from –180° to 180°. Units are in degrees.

More About

Broadside Angle

Broadside angles are useful in describing the response pattern of a uniform linear array (ULA).

For the definition of the broadside angle and how to convert between azimuth and elevation, and broadside angle see “Broadside Angles”. For definitions of the azimuth and elevation angles, see “Azimuth and Elevation Angles”.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`az2broadside` | `uv2azel` | `phitheta2azel`

bw2range

Convert bandwidth to range resolution

Syntax

```
rngres = bw2range(bw)
rngres = bw2range(bw,c)
```

Description

Note The use of `bw2range` is not recommended. Use `bw2rangeres` instead.

`rngres = bw2range(bw)` returns the range resolution of a signal corresponding to its bandwidth. Range resolution gives you the minimum range difference needed to distinguish two targets. The function applies to two-way propagation, as in a monostatic radar system.

`rngres = bw2range(bw,c)` also specifies the signal propagation speed, `c`.

Input Arguments

`bw` — Signal bandwidth
positive scalar | MATLAB array of positive real values

Signal bandwidth, specified as any array of array of positive real values. Units are in hertz.

`c` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value is the output of `physconst('LightSpeed')`. Units are in meters per second.

Data Types: `double`

Output Arguments

`rngres` — Target range resolution
positive scalar | MATLAB array of positive real values

Target range resolution, returned as a scalar or MATLAB array of positive real numbers. The dimensions of `rngres` are the same as those of `bw`. Units are in meters.

Data Types: `double`

Tips

- This function assumes two-way propagation. For one-way propagation, you can find the required range resolution by multiplying the output of this function by 2.

Algorithms

The function computes range resolution from `rngres = c/(2*bw)`.

Version History

Introduced in R2017a

References

[1] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`time2range` | `range2bw` | `range2time` | `phased.FMCWWaveform`

Topics

"Automotive Adaptive Cruise Control Using FMCW Technology" (Radar Toolbox)

bw2rangeres

Convert bandwidth to range resolution

Syntax

```
rangeres = bw2rangeres(bw)
rangeres = bw2rangeres(bw,c)
rangeres = bw2rangeres( ___ ,'RangeBroadening',rb)
```

Description

`rangeres = bw2rangeres(bw)` returns the range resolution `rangeres` corresponding to the signal bandwidth `bw`. Range resolution gives the minimum range difference needed to distinguish two targets. The propagation is assumed to be two-way, as in a monostatic radar system.

`rangeres = bw2rangeres(bw,c)` also specifies the signal propagation speed `c`.

`rangeres = bw2rangeres( ___ ,'RangeBroadening',rb)` also specifies the range broadening factor `rb`.

Examples

Compute Range Resolution from Bandwidth

Assume you have a monostatic radar system that uses a rectangular waveform. Calculate the range resolution obtained using a signal bandwidth of 20 MHz.

```
bw = 20e6;
rngres = bw2rangeres(bw)
```

```
rngres = 7.4948
```

Compute Sonar Range Resolution from Bandwidth

Calculate the range resolution of a two-way sonar system that uses a rectangular waveform. The signal bandwidth is 2 kHz. The speed of sound is 1520 m/s.

```
bw = 2e3;
c = 1520.0;
rngres = bw2rangeres(bw,c)
```

```
rngres = 0.3800
```

Compute Range Resolution from Bandwidth with Range Broadening

Assume a monostatic radar system that uses a rectangular waveform. Calculate the range resolution for two signals with bandwidths of 10 and 20 MHz. Use a range broadening factor of 2.0.

```
bw = [10e6,20e6];
c = physconst('Lightspeed');
rngres = bw2rangeres(bw,c,'RangeBroadening',2.0)
```

```
rngres = 1×2

   29.9792   14.9896
```

Input Arguments

bw — Signal bandwidth
positive scalar

Signal bandwidth, specified as a positive scalar. Units are in Hz.

Example: `300e6`

Data Types: `double`

c — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in m/s.

Example: 3.0e8

Data Types: `double`

rb — Range impulse broadening factor
`1.0` (default) | real positive scalar

Range impulse broadening factor, specified as a real scalar. Range broadening occurs due to data weighting or windowing for side lobe control. Range broadening is the ratio of the actual –3 dB main lobe width with respect to the nominal width. Typical window functions such as `hamming` or `hann` exhibit range broadening in the range of 1 to 1.5.

Example: `2.0`

Data Types: `double`

Output Arguments

`rangeres` — Range resolution
positive scalar

Signal range resolution, returned as a positive scalar. Units are in meters.

Algorithms

The range resolution (in meters) for a rectangular pulse of bandwidth *bw* is *c*/(*bw* × 2), where *c* is the propagation speed. The range resolution broadening factor *b* yields a range resolution of (*c* × *b*)/(*bw* × 2).

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`rangeres2bw`

wavelen2freq

Convert wavelength to frequency

Syntax

```
freq = wavelen2freq(wavelen)
freq = wavelen2freq(wavelen,c)
[freq,c] = wavelen2freq( ___ )
```

Description

`freq = wavelen2freq(wavelen)` converts wavelength `wavelen` to frequency `freq` using the speed of light in vacuum.

`freq = wavelen2freq(wavelen,c)` also specifies the signal propagation speed `c`.

`[freq,c] = wavelen2freq( ___ )` also returns the propagation speed `c` used to compute frequency `freq`. Use this syntax with any of the input arguments in previous syntaxes.

Examples

Calculate Frequencies for Different Wavelengths

Calculate the frequencies corresponding to wavelengths of 1 and 4 cm. Use the default speed of propagation.

```
wavelen = [1e-2,4e-2];
freq = wavelen2freq(wavelen)
```

```
freq = 1×2
10^10 ×

    2.9979    0.7495
```

Obtain Propagation Speed Used for Frequency Conversion

Calculate the frequency corresponding to a wavelength of 2 cm. Obtain the propagation speed used to compute the frequency.

```
wavelen = 0.02;
[freq,c] = wavelen2freq(wavelen,3.0e8)
```

```
freq = 1.5000e+10

c = 300000000
```

Input Arguments

`wavelen` — Signal wavelength
positive scalar | length-*M* vector of positive values

Signal wavelength, specified as a positive scalar or length-*M* vector of positive values. Units are in meters.

`c` — Signal propagation speed
speed of light in vacuum (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value is obtained from `physconst('Lightspeed')`. Units are in m/s.

Example: 3.0e8

Data Types: `double`

Output Arguments

`freq` — Signal frequencies
positive scalar | length-*M* vector of positive values

Signal frequency, specified as a positive scalar or a length-*M* vector of positive values. The size of `freq` equals the size of `wavelen`. Units are in Hz.

Example: `10e6`

Data Types: `double`

`c` — Signal propagation speed
speed of light in vacuum (default) | positive scalar

Signal propagation speed, returned as a positive scalar. Units are in m/s.

Data Types: `double`

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`freq2wavelen`

freq2wavelen

Convert frequency to wavelength

Syntax

```
wavelen = freq2wavelen(freq)
wavelen = freq2wavelen(freq,c)
[wavelen,c] = freq2wavelen( ___ )
```

Description

`wavelen = freq2wavelen(freq)` converts the frequency `freq` to the wavelength `wavelen` using the speed of light in a vacuum.

`wavelen = freq2wavelen(freq,c)` also specifies the signal propagation speed `c`.

`[wavelen,c] = freq2wavelen( ___ )` also returns the propagation speed `c` used to compute wavelength `wavelen`.

Examples

Calculate Wavelengths for Different Frequencies

Calculate the wavelengths corresponding to frequencies of 3 and 4 GHz. Use the default speed of propagation.

```
freq = [3e9,4e9];
lambda = freq2wavelen(freq)
```

```
lambda = 1×2

    0.0999    0.0749
```

Obtain Propagation Speed Used for Wavelength Conversion

Calculate the wavelength corresponding to a frequency of 4 GHz. Obtain the propagation speed used to compute the wavelength.

```
freq = 4e9;
[lambda,c] = freq2wavelen(freq,3.0e8)
```

```
lambda = 0.0750

c = 300000000
```

Input Arguments

`freq` — Signal frequencies
positive scalar | length-*M* vector of positive values

Signal frequency, specified as a positive scalar or a length-*M* vector of positive values. Units are in Hz.

Example: `10e6`

Data Types: `double`

`c` — Signal propagation speed
speed of light in vacuum (default) | positive scalar

Signal propagation speed, specified as a positive scalar. The default value is obtained from `physconst('Lightspeed')`. Units are in m/s.

Example: 3.0e8

Data Types: `double`

Output Arguments

`wavelen` — Signal wavelength
positive scalar | length-*M* vector of positive values

Signal wavelength, returned as a positive scalar or length-*M* vector of positive values. The size of `wavelen` equals the size of `freq`. Units are in meters.

`c` — Signal propagation speed
positive scalar

Signal propagation speed, returned as a positive scalar. Units are in m/s.

Data Types: `double`

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`wavelen2freq`

rangeres2bw

Convert range resolution to bandwidth

Syntax

```
bw = rangeres2bw(rangeres)
bw = rangeres2bw(rangeres,c)
bw = rangeres2bw( ___ ,'RangeBroadening',rb)
```

Description

`bw = rangeres2bw(rangeres)` returns the signal bandwidth `bw` corresponding to the range resolution `rangeres`. Signal bandwidth gives the maximum bandwidth needed to distinguish two targets in range. The propagation is assumed to be two-way, as in a monostatic radar system.

`bw = rangeres2bw(rangeres,c)` also specifies the signal propagation speed `c`.

`bw = rangeres2bw( ___ ,'RangeBroadening',rb)` also specifies the range broadening factor `rb`.

Examples

Compute Bandwidth from Range Resolution

Assume you have a monostatic radar system that uses a rectangular waveform. Calculate the signal bandwidth needed to obtain a range resolution of 25 m. Use the default speed of light.

```
rangeres = 25.0;
bw = rangeres2bw(rangeres)
```

```
bw = 5.9958e+06
```

Compute Sonar Bandwidth from Range Resolution

Calculate the bandwidth of a two-way sonar system that uses a rectangular waveform. The desired range resolution is one meter. The speed of sound is 1520 m/s.

```
rangeres = 1;
c = 1520.0;
bw = rangeres2bw(rangeres,c)
```

```
bw = 760
```

Compute Bandwidth from Range Resolution with Range Broadening

Assume a monostatic radar system that uses a rectangular waveform. Calculate the bandwidths of two signals with range resolutions of 1 and 10 meters. The range broadening factor is 2.0.

```
res = [1.0,10.0];
c = physconst('Lightspeed');
bw = rangeres2bw(res,c,'RangeBroadening',2.0)
```

```
bw = 1×2
10^8 ×

    2.9979    0.2998
```

Input Arguments

rangeres — Range resolution
positive scalar

Range resolution, specified as a positive scalar. Units are in meters.

c — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in m/s.

Example: `3.0e8`

Data Types: `double`

rb — Range impulse broadening factor
`1.0` (default) | real positive scalar

Range impulse broadening factor, specified as a real scalar. Range broadening occurs due to data weighting or windowing for side lobe control. Range broadening is the ratio of the actual –3 dB main lobe width with respect to the nominal width. Typical window functions such as `hamming` or `hann` exhibit range broadening in the range of 1 to 1.5.

Example: `2.0`

Data Types: `double`

Output Arguments

bw — Signal bandwidth
positive scalar

Signal bandwidth, returned as a positive scalar. Units are in Hz.

Example: `300e6`

Data Types: `double`

Algorithms

The range resolution (in meters) for a rectangular pulse of bandwidth, *bw*, is *c/bw/2*, where *c* is the propagation speed. Using the range resolution broadening factor *b* yields a range resolution of *cb/bw/2*.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`bw2rangeres`

ap2beamwidth

Compute half-power beamwidth from aperture length

Syntax

```
rlhpbw = ap2beamwidth(d,lambda)
rlhpbw = ap2beamwidth(d,lambda,azb)
```

Description

`rlhpbw = ap2beamwidth(d,lambda)` computes the half-power, one-way beamwidth, `rlhpbw` for a real aperture. The quantity `d` specifies the length of an unweighted antenna and `lambda` specifies the radar wavelength.

`rlhpbw = ap2beamwidth(d,lambda,azb)` also specifies the azimuth impulse broadening factor `azb`.

Examples

Compute Beamwidths of Multiple Antennas

Estimate the half-power beamwidths of side-looking airborne radars, each operating at wavelengths of 1, 2, and 3 centimeters. The radars have antenna aperture sizes of 2 and 3 meters in the azimuth direction.

```
lambda = [1,2,3]*1e-2;
daz = [2,3];
hpbw = ap2beamwidth(daz,lambda)
```

```
hpbw = 2×3

    0.2865    0.5730    0.8594
    0.1910    0.3820    0.5730
```

Compute Beamwidth of Antenna with Azimuth Broadening

Estimate the half-power beamwidth of a radar operating at a wavelength of 2 cm. The radar has an antenna aperture dimension of 2 m in the azimuth direction. Use an azimuth broadening factor of 1.5.

```
lambda = 2e-2;
daz = 2;
azbf = 1.5;
hpbw = ap2beamwidth(daz,lambda,azbf)
```

```
hpbw = 0.8594
```

Input Arguments

d — Length of unweighted antenna
positive scalar | length-*J* vector of positive values

Length of unweighted antenna, specified as a positive scalar or length-*J* vector of positive values. Units are in meters.

Example: `[3,4][`

Data Types: `double`

`lambda` — Radar wavelength
positive real scalar | length-*K* vector of real values

Radar wavelength, specified as a positive real scalar or a length-*K* vector of real values.

Example: `1.5`

Data Types: `double`

`azb` — Azimuth impulse broadening factor
`1.0` (default) | positive real scalar

Azimuth impulse broadening factor, specified as a positive real scalar. Broadening is due to data weighting or windowing for side lobe control. The quantity is the actual –3 dB main lobe width with respect to the nominal width. Typical window functions such as `hamming` or `hann` exhibit azimuth impulse broadening factors in the range of 1 to 1.5.

Example: `1.2`

Data Types: `double`

Output Arguments

`rlhpbw` — Half-power one-way beamwidth
positive scalar | *J*-by-*K* real-valued vector of positive values

Half-power one-way beamwidth, specified as a positive scalar or as a *J*-by-*K* real-valued vector of positive values. Matrix rows correspond to the length of `d` and columns correspond to the length of `lambda`. Units are in degrees.

Algorithms

The half-power beamwidth (in degrees) for a rectangular aperture is $180\lambda/\pi d$ where λ is the radar wavelength and d is the aperture length. The azimuth broadening factor b multiplies the beamwidth to produce a half-power beamwidth of $180b\lambda/\pi d$.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`beamwidth2ap`

beamwidth2ap

Computes aperture length of unweighted antenna from beamwidth

Syntax

```
d = beamwidth2ap(hpbw,lambda)
d = beamwidth2ap(hpbw,lambda,azb)
```

Description

`d = beamwidth2ap(hpbw,lambda)` computes the aperture length `d` of an unweighted antenna from its half-power one-way beamwidth `hpbw`. `lambda` specifies the radar wavelength.

`d = beamwidth2ap(hpbw,lambda,azb)` also specifies the azimuth impulse broadening factor `azb`.

Examples

Compute Aperture Length of Antenna from Beamwidth

Compute the aperture length of a radar from its half-power beamwidth. The radar is designed to operate at a wavelength of 2 centimeters. The radar has a half-power beamwidth of one degree. Use the default azimuth broadening factor.

```
lambda = 2e-2;
bw = 1;
aplen = beamwidth2ap(bw,lambda)
```

```
aplen = 1.1459
```

Compute Aperture Lengths of Multiple Antennas from Beamwidths

Estimate the aperture lengths of two side-looking airborne radars operating at wavelengths of 1, 2, and 3 centimeters. The radars have beamwidths of 2 and 3 degrees in the azimuth dimension.

```
lambda = [1,2,3]*1e-2;
bw = [2,3];
aplen = beamwidth2ap(bw,lambda)
```

```
aplen = 2×3

    0.2865    0.5730    0.8594
    0.1910    0.3820    0.5730
```

Compute Aperture Length of Antenna with Azimuth Broadening

Compute the aperture length of a radar operating at a wavelength of 2 centimeters. The radar has a beamwidth of 2 deg in the azimuth direction. Use an azimuth broadening factor of 1.5.

```
lambda = 2e-2;
bw = 2;
azbf = 1.5;
ap = beamwidth2ap(bw,lambda,azbf)
```

```
ap = 0.8594
```

Compute Aperture Lengths of Rectangular Antenna

Estimate the aperture length of each side of a rectangular aperture antenna. The antenna operates at a wavelength of 10 cm. In this case, the input beamwidth array `bw` describes the two beamwidths of the same antenna. The radar has a half-power beamwidth of 1 degree along one dimension and a half-power beamwidth of 2 degrees along the other.

```
bw = [1,2];
lambda = 0.01;
ap = beamwidth2ap(bw,lambda)
```

```
ap = 2×1

    0.5730
    0.2865
```

Input Arguments

`hpbw` — Half-power one-way beamwidth
positive scalar | *J*-by-*K* real-valued vector of positive values

Half-power one-way beamwidth, specified as a positive scalar or as a *J*-by-*K* real-valued vector of positive values. Matrix rows correspond to the length of `d` and columns correspond to the length of `lambda`. Units are in degrees.

`lambda` — Radar wavelength
positive real scalar | length-*K* vector of real values

Radar wavelength, specified as a positive real scalar or a length-*K* vector of real values.

Example: `1.5`

Data Types: `double`

`azb` — Azimuth impulse broadening factor
`1.0` (default) | positive real scalar

Azimuth impulse broadening factor, specified as a positive real scalar due to data weighting or windowing for side lobe control. The quantity is the actual –3 dB main lobe width with respect to the nominal width. Typical window functions such as `hamming` or `hann` exhibit azimuth impulse broadening factors in the range of 1 to 1.5.

Example: 1.2

Data Types: double

Output Arguments

d — Length of unweighted antenna
positive scalar | length-*J* vector of positive values

Length of unweighted antenna, returned as a positive scalar or length-*J* vector of positive values. Units are in meters.

Example: [3.5,4.2]

Data Types: double

Algorithms

The half-power beamwidth (in degrees) for a rectangular aperture is $180\lambda/\pi d$ where λ is the radar wavelength and d is the aperture length. The azimuth broadening factor b multiplies the beamwidth to produce a half-power beamwidth of $180b\lambda/\pi d$.

Version History

Introduced in R2021b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`ap2beamwidth`

cart2sphvec

Convert vector from Cartesian components to spherical representation

Syntax

```
vs = cart2sphvec(vr,az,el)
```

Description

`vs = cart2sphvec(vr,az,el)` converts the components of a vector or set of vectors, `vr`, from their representation in a local Cartesian coordinate system to a spherical basis representation contained in `vs`. A spherical basis representation is the set of components of a vector projected into a basis given by $(\hat{\mathbf{e}}_{az}, \hat{\mathbf{e}}_{el}, \hat{\mathbf{e}}_{R})$. The orientation of a spherical basis depends upon its location on the sphere as determined by azimuth, `az`, and elevation, `el`.

Examples

Spherical Representation of Unit Z-Vector

Start with a vector in Cartesian coordinates pointing along the *z*-direction and located at 45° azimuth, 45° elevation. Compute its components with respect to the spherical basis at that point.

```
vr = [0;0;1];
vs = cart2sphvec(vr,45,45)
```

```
vs = 3×1

         0
    0.7071
    0.7071
```

Input Arguments

`vr` — Vector in Cartesian basis representation
3-by-1 column vector | 3-by-N matrix

Vector in Cartesian basis representation specified as a 3-by-1 column vector or 3-by-N matrix. Each column of `vr` contains the three components of a vector in the right-handed Cartesian basis *x,y,x*.

Example: `[4.0; -3.5; 6.3]`

Data Types: `double`
Complex Number Support: Yes

`az` — Azimuth angle
scalar in range [–180,180]

Azimuth angle specified as a scalar in the closed range [–180,180]. Angle units are in degrees. To define the azimuth angle of a point on a sphere, construct a vector from the origin to the point. The azimuth angle is the angle in the *xy*-plane from the positive *x*-axis to the vector's orthogonal projection into the *xy*-plane. As examples, zero azimuth angle and zero elevation angle specify a point on the *x*-axis while an azimuth angle of 90° and an elevation angle of zero specify a point on the *y*-axis.

Example: `45`

Data Types: `double`

`el` — Elevation angle
scalar in range [–90,90]

Elevation angle specified as a scalar in the closed range [–90,90]. Angle units are in degrees. To define the elevation of a point on the sphere, construct a vector from the origin to the point. The elevation angle is the angle from its orthogonal projection into the *xy*-plane to the vector itself. As examples, zero elevation angle defines the equator of the sphere and ±90° elevation define the north and south poles, respectively.

Example: `30`

Data Types: `double`

Output Arguments

`vs` — Vector in spherical basis
3-by-1 column vector | 3-by-N matrix

Spherical representation of a vector returned as a 3-by-1 column vector or 3-by-N matrix having the same dimensions as `vs`. Each column of `vs` contains the three components of the vector in the right-handed $(\hat{\mathbf{e}}_{az}, \hat{\mathbf{e}}_{el}, \hat{\mathbf{e}}_R)$ basis.

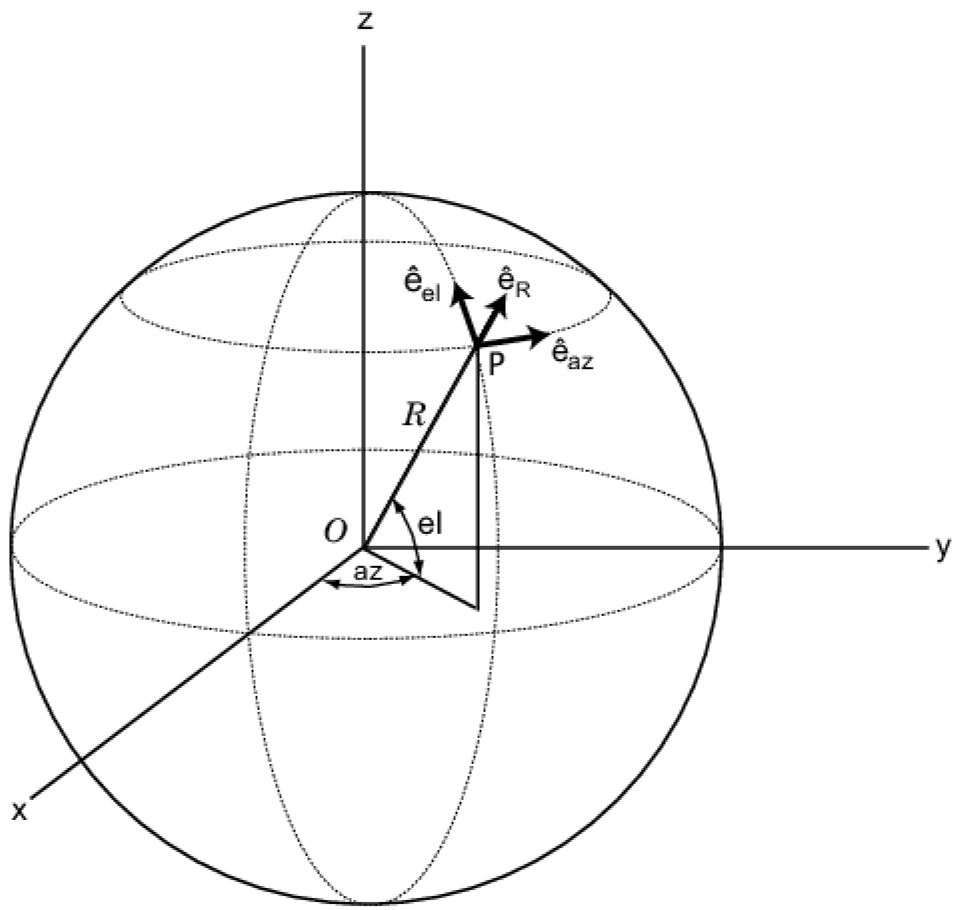
More About

Spherical basis representation of vectors

Spherical basis vectors are a local set of basis vectors which point along the radial and angular directions at any point in space.

The spherical basis is a set of three mutually orthogonal unit vectors $(\hat{\mathbf{e}}_{az}, \hat{\mathbf{e}}_{el}, \hat{\mathbf{e}}_R)$ defined at a point on the sphere. The first unit vector points along lines of azimuth at constant radius and elevation. The second points along the lines of elevation at constant azimuth and radius. Both are tangent to the surface of the sphere. The third unit vector points radially outward.

The orientation of the basis changes from point to point on the sphere but is independent of *R* so as you move out along the radius, the basis orientation stays the same. The following figure illustrates the orientation of the spherical basis vectors as a function of azimuth and elevation:



For any point on the sphere specified by *az* and *el*, the basis vectors are given by:

$$\hat{\mathbf{e}}_{\mathbf{az}} = -\sin(az)\hat{\mathbf{i}} + \cos(az)\hat{\mathbf{j}}$$

$$\hat{\mathbf{e}}_{\mathbf{el}} = -\sin(el)\cos(az)\hat{\mathbf{i}} - \sin(el)\sin(az)\hat{\mathbf{j}} + \cos(el)\hat{\mathbf{k}}$$

$$\hat{\mathbf{e}}_{\mathbf{R}} = \cos(el)\cos(az)\hat{\mathbf{i}} + \cos(el)\sin(az)\hat{\mathbf{j}} + \sin(el)\hat{\mathbf{k}} \quad .$$

Any vector can be written in terms of components in this basis as $\mathbf{v} = v_{az}\hat{\mathbf{e}}_{\mathbf{az}} + v_{el}\hat{\mathbf{e}}_{\mathbf{el}} + v_R\hat{\mathbf{e}}_{\mathbf{R}}$. The transformations between spherical basis components and Cartesian components take the form

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} -\sin(az) & -\sin(el)\cos(az) & \cos(el)\cos(az) \\ \cos(az) & -\sin(el)\sin(az) & \cos(el)\sin(az) \\ 0 & \cos(el) & \sin(el) \end{bmatrix} \begin{bmatrix} v_{az} \\ v_{el} \\ v_R \end{bmatrix}$$

.

and

$$\begin{bmatrix} v_{az} \\ v_{el} \\ v_R \end{bmatrix} = \begin{bmatrix} -\sin(az) & \cos(az) & 0 \\ -\sin(el)\cos(az) & -\sin(el)\sin(az) & \cos(el) \\ \cos(el)\cos(az) & \cos(el)\sin(az) & \sin(el) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}.$$

Version History

Introduced in R2013a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`sph2cartvec` | `azelaxes`

cbfweights

Conventional beamformer weights

Syntax

```
wt = cbfweights(pos,ang)
wt = cbfweights(pos,ang,nqbits)
```

Description

`wt = cbfweights(pos,ang)` returns narrowband conventional beamformer weights. When applied to the elements of a sensor array, these weights steer the response of the array to a specified arrival direction or set of directions. The `pos` argument specifies the sensor positions in the array. The `ang` argument specifies the azimuth and elevation angles of the desired response directions. The output weights, `wt`, are returned as an *N*-by-*M* matrix. In this matrix, *N* represents the number of sensors in the array while *M* represents the number of arrival directions. Each column of `wt` contains the weights for the corresponding direction specified in the `ang`. The argument `wt` is equivalent to the output of the function `steervec` divided by *N*. All elements in the sensor array are assumed to be isotropic.

`wt = cbfweights(pos,ang,nqbits)` returns quantized narrowband conventional beamformer weights when the number of phase-shifter bits is set to `nqbits`.

Examples

Conventional Weights for Two Beamformer Directions

Specify a line array of five elements spaced 10 cm apart. Compute the weights for two directions: 30° azimuth, 0° elevation, and 45° azimuth, 0° elevation. Assume the array is tuned to plane waves having a frequency of 1 GHz.

```
elementPos = (0:.1:.4);
c = physconst('LightSpeed');
fc = 1e9;
lambda = c/fc;
ang = [30 45];
wt = cbfweights(elementPos/lambda,ang)
```

```
wt = 5×2 complex

   0.2000 + 0.0000i   0.2000 + 0.0000i
   0.0999 + 0.1733i   0.0177 + 0.1992i
  -0.1003 + 0.1731i  -0.1969 + 0.0353i
  -0.2000 - 0.0004i  -0.0527 - 0.1929i
  -0.0995 - 0.1735i   0.1875 - 0.0696i
```

Quantized Weights for Two Beamformer Directions

Specify a line array of five elements spaced 10 cm apart. Compute the weights for two directions: 30° azimuth, 0° elevation, and 45° azimuth, 0° elevation. Assume the array is tuned to plane waves having a frequency of 1 GHz. Assume the weights are quantized to six bits.

```
elementPos = (0:.1:.4);
c = physconst('LightSpeed');
fc = 1e9;
lambda = c/fc;
ang = [30 45];
nqbits = 6;
wt = cbfweights(elementPos/lambda,ang,nqbits)
```

```
wt = 5×2 complex

   0.2000 + 0.0000i   0.2000 + 0.0000i
   0.0943 + 0.1764i   0.0196 + 0.1990i
  -0.0943 + 0.1764i  -0.1962 + 0.0390i
  -0.2000 + 0.0000i  -0.0581 - 0.1914i
  -0.0943 - 0.1764i   0.1848 - 0.0765i
```

Input Arguments

pos — Positions of array sensor elements
1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array, specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero. When `pos` is a 2-by-*N* matrix, it represents the *(y,z)*-coordinates of the sensor elements of a planar array. This array is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. When `pos` is a 3-by-*N* matrix, then the array can have an arbitrary shape. Sensor positions are in terms of signal wavelength.

Example: `[0,0,0; 0.1,0.4,0.3; 1,1,1]`

Data Types: `double`

ang — Beamforming directions
1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Beamforming directions, specified as a 1-by-*M* vector or a 2-by-*M* matrix. In this vector or matrix, *M* represents the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the beamforming direction as `[az;el]`. Angular units are specified in degrees. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the beamforming direction vector onto the *xy* plane. The angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the beamforming direction vector and *xy*-plane. It is positive when measured towards the positive *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero.

Example: `[45;10]`

Data Types: `double`

`nqbits` — Number of phase shifter quantization bits
0 (default) | non-negative integer

Number of bits used to quantize the phase shift in beamformer or steering vector weights, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Example: 5

Output Arguments

`wt` — Beamformer weights
N-by-*M* complex-valued matrix

Beamformer weights returned as an *N*-by-*M* complex-valued matrix. In this matrix, *N* represents the number of sensor elements of the array while *M* represents the number of beamforming directions. Each column of `wt` corresponds to a beamforming direction specified in `ang`.

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York, NY: Wiley-Interscience, 2002.

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. "Beamforming: A versatile approach to spatial filtering". *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`lcmvweights` | `mvdrweights` | `sensorcov` | `steervec` | `phased.PhaseShiftBeamformer`

circpol2pol

Convert circular component representation of field to linear component representation

Syntax

```
fv = circpol2pol(cfv)
```

Description

`fv = circpol2pol(cfv)` converts the circular polarization components of the field or fields contained in `cfv` to their linear polarization components contained in `fv`. Any polarized field can be expressed as a linear combination of horizontal and vertical components.

Examples

Convert Circular to Linear Polarization

Convert a horizontally polarized field, originally expressed in circular polarization components, into linear polarization components.

```
cfv = [1;1];
fv = circpol2pol(cfv)
```

```
fv = 2×1

    1.4142
         0
```

The vertical component of the output is zero for horizontally polarized fields.

Convert Circular Polarization Ratio to Linear Polarization Ratio

Create a right circularly polarized field. Compute the circular polarization ratio and convert to a linear polarization ratio equivalent. Note that the input circular polarization ratio is `Inf`.

```
cfv = [0;1];
q = cfv(2)/cfv(1);
p = circpol2pol(q)
```

```
p = 0.0000 - 1.0000i
```

Input Arguments

`cfv` — Field vector in circular polarization representation
1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix

Field vector in its circular polarization representation specified as a 1-by-*N* complex row vector or a 2-by-*N* complex matrix. If `cfv` is a matrix, each column represents a field in the form of `[El;Er]`, where `El` and `Er` are the left and right circular polarization components of the field. If `cfv` is a row vector, each column in `cfv` represents the polarization ratio, `Er/El`. For a row vector, the value `Inf` can designate the case when the ratio is computed for `El = 0`.

Example: [1;-1]

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`fv` — Field vector in linear polarization representation or Jones vector
1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix

Field vector in linear polarization representation or Jones vector returned as a 1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix. `fv` has the same dimensions as `cfv`. If `cfv` is a matrix, each column of `fv` contains the horizontal and vertical linear polarization components of the field in the form, `[Eh;Ev]`. If `cfv` is a row vector, each entry in `fv` contains the linear polarization ratio, defined as `Ev/Eh`.

Version History

Introduced in R2013a

References

[1] Mott, H., *Antennas for Radar and Communications*, John Wiley & Sons, 1992.

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`pol2circpol` | `polellip` | `polratio` | `stokes`

coincidence

Coincidence algorithm

Syntax

```
x = coincidence(res,div,maxval)
x = coincidence(res,div,maxval,tol)
```

Description

`x = coincidence(res,div,maxval)` returns the scalar `x` that is less than or equal to `maxval` and is congruent to each remainder in `res` for the corresponding divisor in `div`. `x` satisfies

mod(x,div) = `res`.

In other words, dividing `x` by each element of `div` leaves as remainder the corresponding element of `res`.

`x = coincidence(res,div,maxval,tol)` also specifies the tolerance. In practice, there may be no value that satisfies all constraints in `res` and `div` exactly. In that case, `coincidence` identifies a set of candidates that approximately satisfy the constraints and are within an interval of width 2 × `tol` centered at the candidates' median. The function then returns the median as `x`.

Examples

Coincidence Algorithm

Find a number smaller than `1000` that has a remainder of about `12` when divided by `19`, a remainder of about `13.1` when divided by `20.4`, and a remainder of about `6.1` when divided by `11`.

There is no number that satisfies the constraints exactly, so specify a tolerance of `1`.

```
rems = [12 13.1 6.1];
divs = [19 20.4 11];

tol = 1;

x = coincidence(rems,divs,1000,tol)
```

```
x = 809.1000
```

Verify that the true remainders are within the specified tolerance.

```
tr = x - floor(x./divs).*divs
```

```
tr = 1×3

   11.1000   13.5000    6.1000
```

Repeat the computation, but now specify a tolerance of `3`. The number that satisfies the constraints decreases as the tolerance increases.

```
tol = 3;
```

```
x = coincidence(rems,divs,1000,tol)
```

```
x = 31
```

```
tr = x - floor(x./divs).*divs
```

```
tr = 1×3

   12.0000   10.6000    9.0000
```

Increase the tolerance to 6. The tolerance has to be smaller than the smallest specified remainder.

```
tol = 6;
```

```
x = coincidence(rems,divs,1000,tol)
```

```
x = 12
```

```
tr = x - floor(x./divs).*divs
```

```
tr = 1×3

    12    12     1
```

Staggered PRF Radar with Maximum Range

In a staggered pulse repetition frequency (PRF) radar system, the first PRF corresponds to 70 range bins and the second PRF corresponds to 85 range bins. The target is detected at bin 47 for the first PRF and bin 12 for the second PRF. Assuming each range bin is 50 meters, compute the target range from these two measurements. Assume the farthest target can be 50 km away.

```
idx = coincidence([47 12],[70 85],50e3/50);
r = 50*idx
```

```
r = 30350
```

Input Arguments

res — Remainder array
row vector of nonnegative numbers

Remainder array, specified as a row vector of nonnegative numbers. `res` must have the same number of elements as `div`.

Data Types: `single` | `double`

div — Divisor array
row vector of positive numbers

Divisor array, specified as a row vector of positive numbers. `div` must have the same number of elements as `res`.

Data Types: single | double

maxval — Upper bound
positive scalar

Upper bound, specified as a positive scalar.

Data Types: single | double

tol — Tolerance
0 (default) | nonnegative scalar

Tolerance, specified as a nonnegative scalar. tol must be smaller than the smallest element of res.

Data Types: single | double

Output Arguments

x — Congruent value
scalar

Congruent value, returned as a scalar.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

crt | iscoprime

cranerainpl

RF signal attenuation due to rainfall using Crane model

Syntax

```
L = cranerainpl(range,freq,rainrate)
L = cranerainpl(range,freq,rainrate,elev)
L = cranerainpl(range,freq,rainrate,elev,tau)
```

Description

`L = cranerainpl(range,freq,rainrate)` returns the signal attenuation, `L`, due to rain based on the Crane rain model [1]. Signal attenuation is a function of the signal path length, `range`, the signal frequency, `freq`, and the rain rate, `rainrate`. The rain rate is defined as the long-term statistical rain rate. The attenuation model applies only for frequencies from 1 GHz to 1000 GHz and is valid for ranges up to 22.5 km. The Crane model accounts for the cellular nature of rainstorms.

`L = cranerainpl(range,freq,rainrate,elev)` also specifies the elevation angle, `elev`, of the signal path.

`L = cranerainpl(range,freq,rainrate,elev,tau)` also specifies the polarization tilt angle, `tau`, of the signal.

Examples

Compare Attenuation for Two Rain Rates Using Crane Model

Use the Crane rain model to compute the signal attenuation caused by rain for a 20 GHz signal sent over a distance of 10 km. Use rain rates of 10.0 and 100.0 mm/hr.

First, set the rain rate to 10 mm/hr.

```
rr = 10.0;
L = cranerainpl(10e3,20.0e9,rr)
```

```
L = 12.5988
```

Repeat the computation using a rain rate of 100.0 mm/hr.

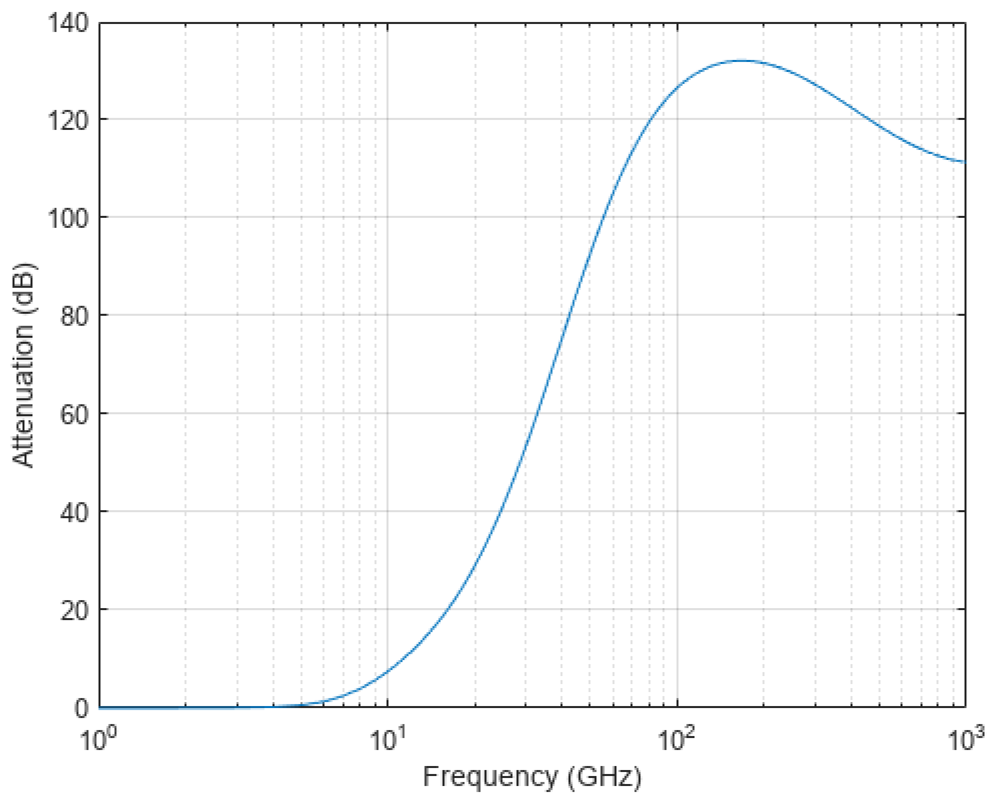
```
rr = 100.0;
L = cranerainpl(10e3,20.0e9,rr)
```

```
L = 73.1912
```

Rain Attenuation as a Function of Frequency Using Crane Model

Plot the signal attenuation due to rain for signals in the frequency range from 1 to 1000 GHz. Use the Crane model to compute the attenuation for a rain rate of 30.0 mm/hr and a signal path distance of 10 km.

```
rr = 30.0;
freq = [1:1000]*1e9;
L = cranerainpl(10e3,freq,rr);
semilogx(freq/1e9,L)
grid
xlabel('Frequency (GHz)')
ylabel('Attenuation (dB)')
```



Rain Attenuation as a Function of Elevation Using Crane Model

Plot the signal attenuation due to rain as a function of elevation angle. Elevation angles vary from 0 to 90 degrees. Assume a path distance of 10 km and a signal frequency of 10 GHz. The rain rate is 100 mm/hr.

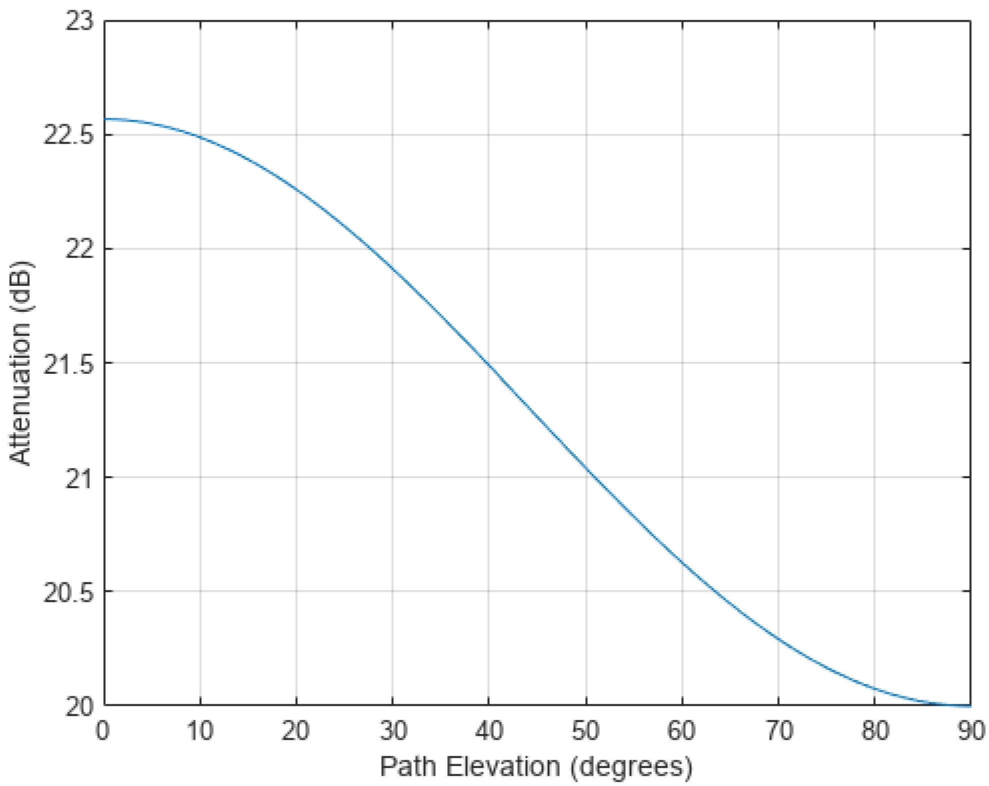
```
rr = 100.0;
```

Set the elevation angles, frequency, and path length.

```
elev = [0:1:90];
freq = 10.0e9;
rng = 10e3*ones(size(elev));
```

Compute and plot the loss.

```
L = cranerainpl(rng,freq,rr,elev);
plot(elev,L)
grid
xlabel('Path Elevation (degrees)')
ylabel('Attenuation (dB)')
```



Rain Attenuation as a Function of Polarization Using Crane Model

Plot the signal attenuation due to rainfall as a function of the polarization tilt angle. Assume a path distance of 10 km, a signal frequency of 10 GHz, and a path elevation angle of 0 degrees. Set the rainfall rate to 70 mm/hour. Plot the signal attenuation against polarization tilt angle.

Set the polarization tilt angle to vary from -90 to 90 degrees.

```
tau = -90:90;
```

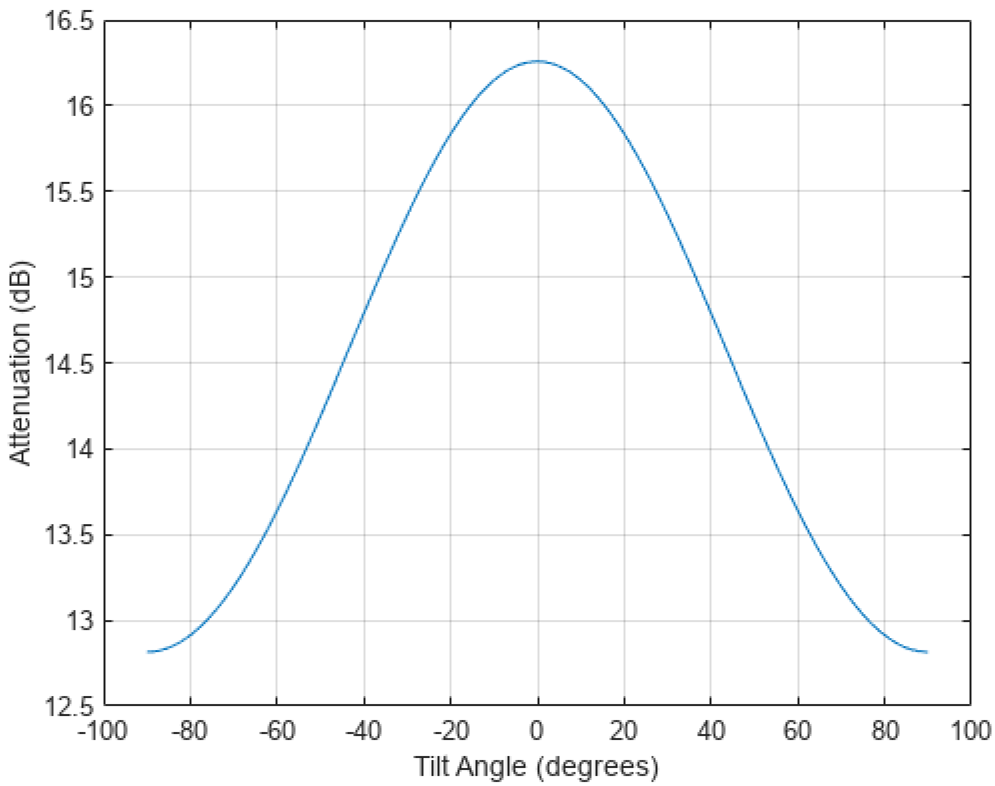
Set the elevation angle, frequency, path distance, and rain rate.

```
elev = 0;
freq = 10.0e9;
```

```
rng = 10e3*ones(size(tau));
rr = 70.0;
```

Compute and plot the attenuation.

```
L = cranerainpl(rng,freq,rr,elev,tau);
plot(tau,L)
grid
xlabel('Tilt Angle (degrees)')
ylabel('Attenuation (dB)')
```



Input Arguments

range — Signal path length

positive scalar | real-valued 1-by-M vector of positive values | real-valued M-by-1 vector of positive values

Signal path length, specified as a positive scalar, a real-valued 1-by-M vector of positive values, or real-valued M-by-1 vector of positive values. Units are in meters.

Example: `[13000.0,14000.0]`

freq — Signal frequency

positive scalar | real-valued 1-by-N vector of positive values | real-valued N-by-1 vector of positive values

Signal frequency, specified as a positive scalar, a real-valued 1-by-*N* vector of positive values, or a real-valued *N*-by-1 vector of positive values. Units are in Hz. Frequencies must lie in the range 1–1000 GHz.

Example: `[2.0:2:10.0]*1e9]`

`rainrate` — Rain rate

nonnegative scalar

Rain rate, specified as a nonnegative scalar. Rain rate represents the long-term statistical rainfall rate provided by Crane (see [1]). Units are in mm/hr.

Example: `100.5`

`elev` — Signal path elevation angle

0.0 (default) | scalar | real-valued 1-by-*M* vector | real-valued *M*-by-1 vector

Signal path elevation angle, specified as a real-valued scalar, or real-valued *M*-by-1 or real-valued 1-by-*M* vector. Units are in degrees between –90° and 90°.

- If `elev` is a scalar, all propagation paths have the same elevation angle.
- If `elev` is a vector, its length must match the length of `range` and each element in `elev` corresponds to a propagation range.

Example: `[0,45]`

`tau` — Tilt angle of signal polarization ellipse

0.0 (default) | scalar | real-valued 1-by-*M* vector | real-valued *M*-by-1 vector

Tilt angle of the signal polarization ellipse, specified as a scalar, a real-valued 1-by-*M* vector, or a real-valued *M*-by-1 vector. Tilt angle values are in the range –90° and 90°, inclusive. Units are in degrees.

- If `tau` is a scalar, all signals have the same tilt angle.
- If `tau` is a vector, its length must match the length of `range`. In that case, each element in `tau` corresponds to a propagation path in `range`.

The tilt angle is defined as the angle between the semimajor axis of the polarization ellipse and the *x*-axis. Because the ellipse is symmetrical, a tilt angle of 10° corresponds to the same polarization state as a tilt angle of -80°. Thus, the tilt angle need only be specified between ±90°.

Example: `[45,30]`

Output Arguments

`L` — Signal attenuation

real-valued *M*-by-*N* matrix

Signal attenuation, returned as a real-valued *M*-by-*N* matrix. Each matrix row represents a different path where *M* is the number of paths. Each column represents a different frequency where *N* is the number of frequencies. Units are in dB.

More About

Crane Rainfall Attenuation Model

The Crane model calculates the attenuation of signals that propagate through regions of rainfall. The model was developed for use on Earth–space or terrestrial propagation paths and is a commonly-used method for the calculation of rain attenuation. The model is based on observations of rain rate, rain structure, and the vertical variation of temperature in the atmosphere. The Crane model (see *Electromagnetic Wave Propagation through Rain*) is primarily applicable to North America. The Crane model generally predicts losses greater than those of the ITU rain attenuation model used in the `rainpl` function. However, the uncertainty of both models and the short-term variation of fade can be large.

The ITU and Crane models are very similar but have some differences. The ITU and Crane rain attenuation models both require statistical annual rainfall rates and utilize an effective path length reduction factor to account for the cellular nature of storms. The 0.01% rainfall rate tables provided by Crane and the ITU are different. The Crane rainfall zones are similar to the ITU zones but more zones are defined in the US than in the ITU model. The ITU rainfall zones are discussed in *ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. The Crane model is more complex consisting of a piecewise combination of path profiles composed of exponential functions.

The Crane model utilizes two exponential functions to span the distance from 0 to 22.5 km.

- For $\delta < D < 22.5$,

$$L = \gamma\left(\frac{e^{y\delta} - 1}{y} - \frac{b^{\alpha}e^{z\delta}}{z} + \frac{b^{\alpha}e^{zD}}{z}\right)$$

- For $0 < D < \delta$,

$$L = \gamma\left(\frac{e^{yD} - 1}{y}\right)$$

where

- L = path attenuation (dB)
- □□= propagation distance (km)
- R = statistical 0.01% rain rate (mm/hr)
- γ = specific attenuation identical to that calculated in `rainpl`.

$$\gamma_R = kR^{\alpha},$$

The parameters k and α depend on the frequency, the polarization state, and the elevation angle of the signal path. These coefficients, given by both Crane *Electromagnetic Wave Propagation through Rain* and the *ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*, are identical and are valid from 1 GHz to 1000 GHz. The specific attenuation model is valid for frequencies from 1–1000 GHz. Rainfall specific attenuation is computed according to the ITU rainfall model in *ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*.

The remaining parameters are empirical constants defined as:

- $b = 2.3R^{-0.17}$

- c = 0.026 - 0.03ln R
- δ = 3.8 - 0.6 ln R
- u = ln $(be^{c\delta})/\delta$
- $y = \alpha u$
- $z = \alpha c$

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the propagation distance.

You can also apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Version History

Introduced in R2020a

References

[1] Crane, Robert K. *Electromagnetic Wave Propagation through Rain*. Wiley, 1996.

[2] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. P Series, Radiowave Propagation 2005.

[3] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.530-17: Propagation data and prediction methods required for the design of terrestrial line-of-sight systems*. 2017.

[4] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.837-7: Characteristics of precipitation for propagation modelling*. 6/2017

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`fspl` | `gaspl` | `fogpl` | `rainpl` | `LOSChannel` | `WidebandLOSChannel`

crt

Chinese remainder theorem

Syntax

```
x = crt(res,div)
```

Description

`x = crt(res,div)` returns the scalar integer `x` that is congruent to each remainder in `res` for the corresponding divisor in `div`. `x` satisfies

mod(x,`div`) = `res`.

In other words, dividing `x` by each element of `div` leaves as remainder the corresponding element of `res`.

Examples

Chinese Remainder Theorem

Find a number that has a remainder of 2 when divided by 9, a remainder of 3 when divided by 10, and a remainder of 6 when divided by 11.

```
x = crt([2 3 6],[9 10 11])
```

```
x = 83
```

Use the `mod` function to verify the result.

```
ver = mod(x,[9 10 11])
```

```
ver = 1×3

     2     3     6
```

Staggered PRF Radar

In a staggered pulse repetition frequency (PRF) radar system, the first PRF corresponds to 70 range bins and the second PRF corresponds to 85 range bins. The target is detected at bin 47 for the first PRF and bin 12 for the second PRF. Assuming each range bin is 50 meters, compute the target range from these two measurements.

```
idx = crt([47 12],[70 85]);
r = 50*idx
```

```
r = 30350
```

Input Arguments

`res` — Remainder array
row vector of nonnegative integers

Remainder array, specified as a row vector of nonnegative integers. `res` must have the same number of elements as `div`.

Data Types: `single` | `double`

`div` — Divisor array
row vector of positive integers

Divisor array, specified as a row vector of positive integers. `div` must have the same number of elements as `res`.

Data Types: `single` | `double`

Output Arguments

`x` — Congruent integer
scalar

Congruent integer, returned as a scalar.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`coincidence` | `iscoprime`

dechirp

Perform dechirp operation on FMCW signal

Syntax

```
y = dechirp(x,xref)
```

Description

`y = dechirp(x,xref)` mixes the incoming signal, `x`, with the reference signal, `xref`. The signals can be complex baseband signals. In an FMCW radar system, `x` is the received signal and `xref` is the transmitted signal.

Examples

Dechirp FMCW Signal

Dechirp a delayed FMCW signal, and plot the spectrum before and after dechirping.

Create an FMCW signal.

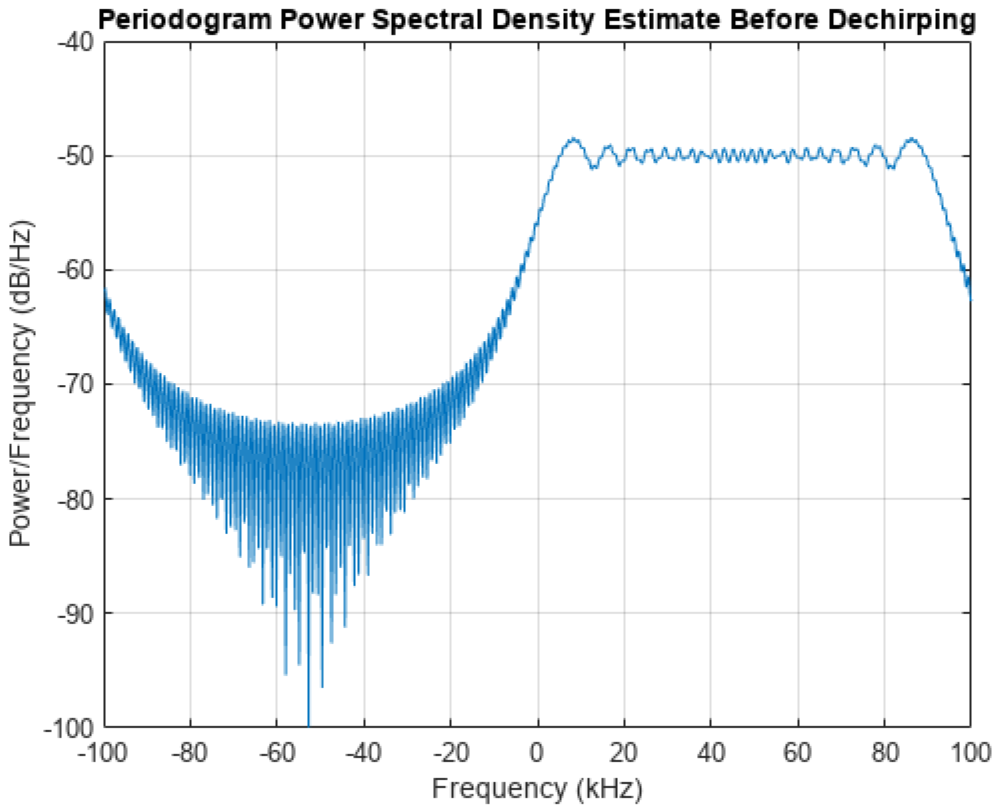
```
Fs = 2e5; Tm = 0.001;
hwav = phased.FMCWWaveform('SampleRate',Fs,'SweepTime',Tm);
xref = step(hwav);
```

Dechirp a delayed copy of the signal.

```
x = [zeros(10,1); xref(1:end-10)];
y = dechirp(x,xref);
```

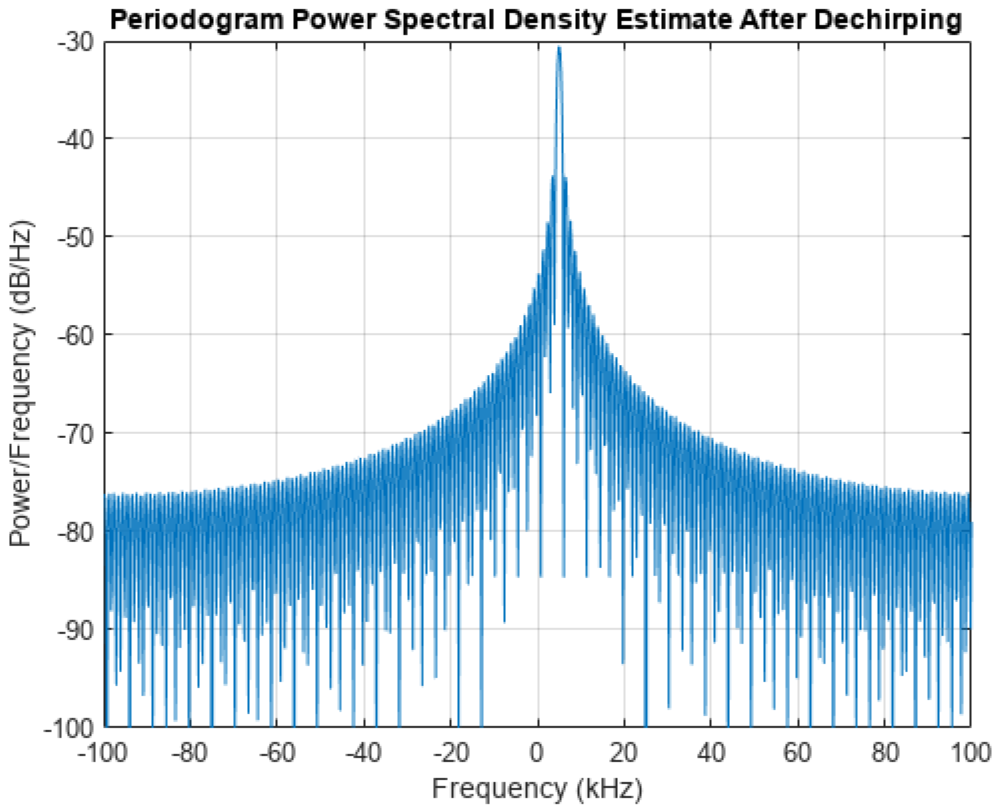
Plot the spectrum before dechirping.

```
[Pxx,F] = periodogram(x,[],1024,Fs,'centered');
plot(F/1000,10*log10(Pxx)); grid;
xlabel('Frequency (kHz)');
ylabel('Power/Frequency (dB/Hz)');
title('Periodogram Power Spectral Density Estimate Before Dechirping');
```



Plot the spectrum after dechirping.

```
[Pyy,F] = periodogram(y,[],1024,Fs,'centered');
plot(F/1000,10*log10(Pyy));
xlabel('Frequency (kHz)');
ylabel('Power/Frequency (dB/Hz)');
ylim([-100 -30]); grid
title('Periodogram Power Spectral Density Estimate After Dechirping');
```



Input Arguments

x — Incoming signal
M-by-*N* matrix

Incoming signal, specified as an *M*-by-*N* matrix. Each column of `x` is an independent signal and is individually mixed with `xref`.

Data Types: `single` | `double`
Complex Number Support: Yes

`xref` — Reference signal
M-by-1 vector

Reference signal, specified as an *M*-by-1 vector.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

y — Dechirped signal
M-by-*N* matrix

Dechirped signal, returned as an *M*-by-*N* matrix. Each column is the mixer output for the corresponding column of `x`.

Data Types: `single` | `double`

Algorithms

For column vectors `x` and `xref`, the mix operation is defined as `xref .* conj(x)`.

If `x` has multiple columns, the mix operation applies the preceding expression to each column of `x` independently.

The mix operation reverses the Doppler shift embedded in `x`, because of the mixing order of `xref` and `x`. The mixing order affects the sign of the imaginary part of the output argument, `y`. There is no consistent convention in the literature about the mixing order. This function and the `beat2range` function use the same convention. If your program processes the output of `dechirp` in other ways, take the mixing order into account.

This function supports single and double precision for input data and arguments. If the input data, `x`, is single precision, the output data is single precision, regardless of the precision of the arguments. If the input data is double precision, the output data is double precision, regardless of the precision of the arguments.

Version History

Introduced in R2012b

References

[1] Pace, Phillip. *Detecting and Classifying Low Probability of Intercept Radar*. Boston: Artech House, 2009.

[2] Skolnik, M.I. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- This function does not support variable-size inputs.
- This function supports single and double precision for input data and arguments. If the input data, `x`, is single precision, the output data is single precision, regardless of the precision of the arguments. If the input data is double precision, the output data is double precision, regardless of the precision of the arguments.

See Also

`beat2range` | `phased.RangeDopplerResponse`

Topics

"Automotive Adaptive Cruise Control Using FMCW Technology" (Radar Toolbox)

delayseq

Delay or advance sequence

Syntax

```
shifted_data = delayseq(data,delay)
shifted_data = delayseq(data,delay,fs)
```

Description

`shifted_data = delayseq(data,delay)` delays or advances the signal in `data` by the number of samples specified in `delay`. Positive values of `delay` delay the signal, while negative values advance the signal. Noninteger values of `delay` represent fractional delays or advances. For fractional delays, the function interpolates between samples.

How the `delayseq` function operates on `data` depends on the dimensions of the `data` and `delay` arguments:

- When `delay` is a scalar, the function applies the same delay to each column of `data`.
- When `delay` is a vector:
  - If `data` is a matrix, the length of the `delay` vector must equal the number of columns in the matrix. The function applies a delay to each column using the corresponding `delay` entry.
  - If `data` is a column vector, the function creates a matrix where each column is the shift in the `data` vector by each entry in `delay`. The number of columns in `shifted_data` equals the length of the `delay` vector. The k^{th} column of `shifted_data` is the result of shifting `data` by `delay(k)`.

`shifted_data = delayseq(data,delay,fs)` specifies `delay` in seconds. `fs` is the sampling frequency of `data`. If the product of `delay` and `fs` is not an integer, `delayseq` implements a fractional delay or advance of the signal using interpolation.

Examples

Delay Signal by Integer Number of Samples

Delay a 1 kHz cosine signal by an integer number of samples. Assume a sampling rate of 10 kHz.

```
fs = 1.0e4;
t = 0:1/fs:0.005;
signal = cos(2*pi*1000*t)';
```

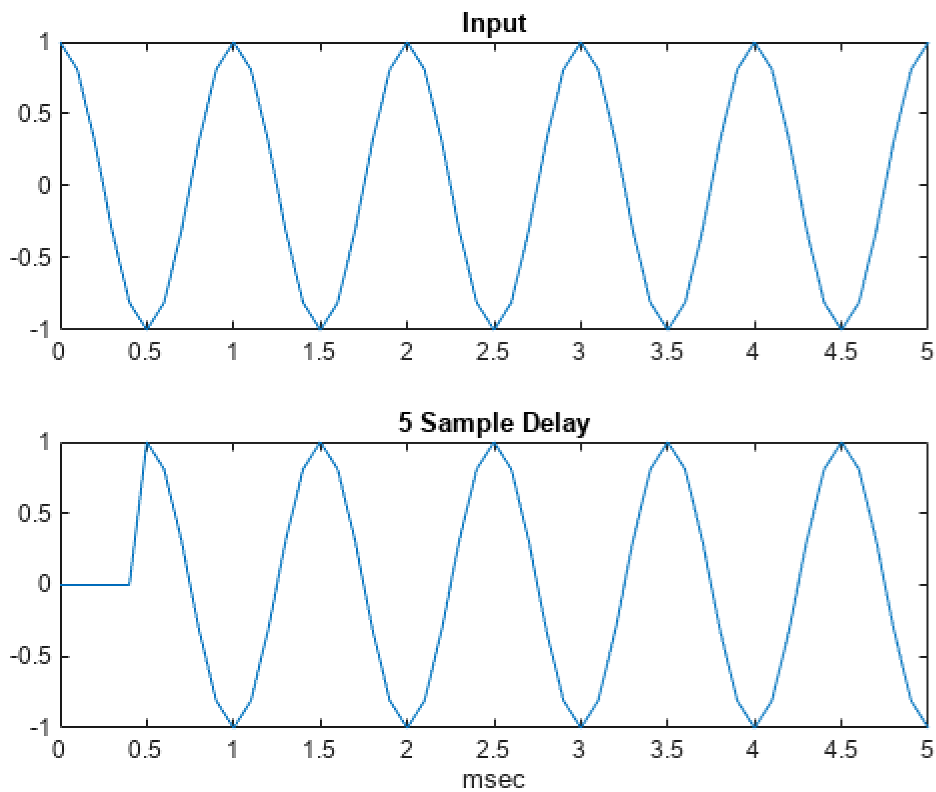
Set the delay to 5 samples (0.5 ms).

```
shifted_signal = delayseq(signal,5);
```

Plot the original and delayed signals.

```
subplot(2,1,1)
plot(t.*1000,signal)
```

```
title('Input')
subplot(2,1,2)
plot(t.*1000,shifted_signal)
title('5 Sample Delay')
xlabel('msec')
```



Delay Signal by Fractional Number of Samples

Delay a 1 kHz cosine signal by a fractional number of samples. Assume a sampling rate of 10 kHz.

```
fs = 1e4;
t = 0:1/fs:0.005;
signal = cos(2*pi*1000*t)';
```

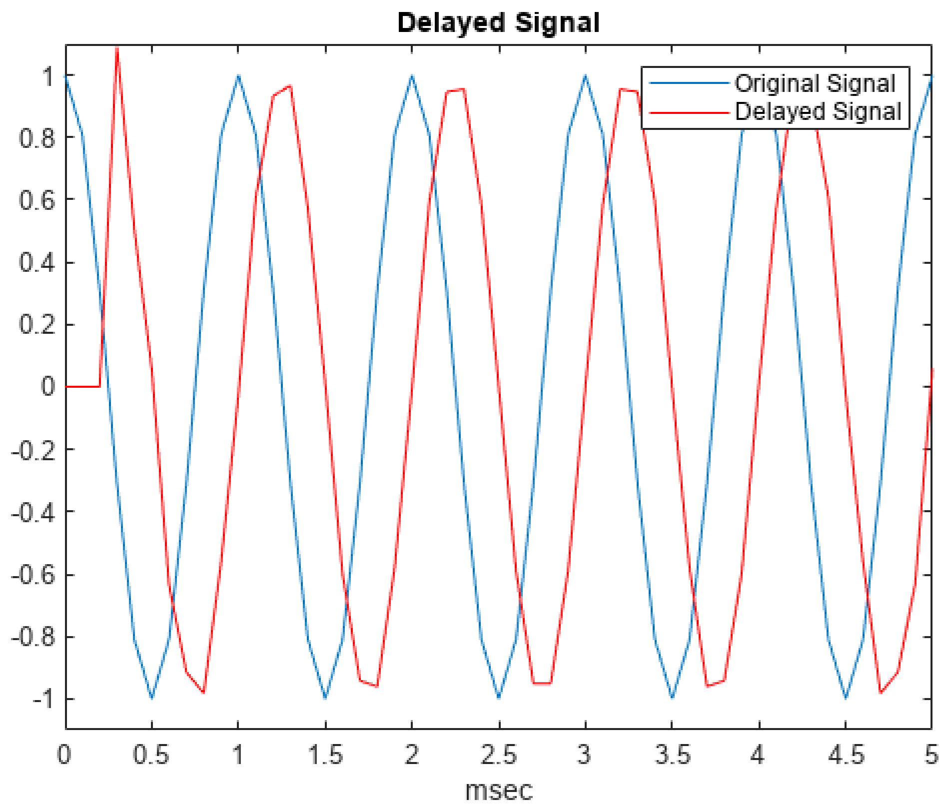
Set the delay to 0.25 ms or 2.5 samples.

```
delayed_signal = delayseq(signal,0.25e-3,fs);
```

Plot the original and delayed signals.

```
plot(t.*1000,signal)
title('Delayed Signal')
hold on
plot(t.*1000,delayed_signal,'r')
axis([0 5 -1.1 1.1])
```

```
xlabel('msec')
legend('Original Signal','Delayed Signal')
hold off
```



The delayed signal values differ from the original signal values because interpolation is used to implement the fractional delay.

Input Arguments

data — Input signal

real-valued length-*M* vector | complex-valued length-*M* vector | real-valued *M*-by-*N* matrix | complex-valued *M*-by-*N* matrix

Input signal, specified as a real-valued length-*M* vector, complex-valued length-*M* vector, real-valued *M*-by-*N* matrix, or complex-valued *M*-by-*N* matrix.

M is the number of samples in `data`. When `data` is a matrix, *N* is the number of independent signals.

Data Types: `single` | `double`
Complex Number Support: Yes

delay — Signal delay or advance

scalar | real-valued *N*-length vector

Signal delay or advance, specified as a scalar or real-value N-length vector. If you specify the fs argument, delay units are in seconds. When delay is a scalar, the same delay is applied to all columns of data. delay units are in samples if fs is not specified and in seconds if fs is specified.

Data Types: single | double

fs — Sampling frequency
1 (default) | positive scalar

Sampling frequency of the signal, specified as a positive scalar. Units are in Hz.

Data Types: single | double

Output Arguments

shifted_data — Delayed or advanced signal
real-valued length-M vector | complex-valued length-M vector | real-valued M-by-N matrix | complex-valued M-by-N matrix

Delayed or advanced signal, returned as a real-valued length-M vector, complex-valued length-M vector, real-valued M-by-N matrix, or complex-valued M-by-N matrix. shifted_data has the same number of rows as data, with appropriate truncations or zero padding.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

phased.TimeDelayBeamformer

diagbfweights

Diagonalize MIMO channel

Syntax

```
[wp,wc] = diagbfweights(chanmat)
[wp,wc,P] = diagbfweights(chanmat)
[wp,wc,P,G] = diagbfweights(chanmat)
[wp,wc,P,G,C] = diagbfweights(chanmat)
[ ___ ] = diagbfweights(chanmat,Pt)
[ ___ ] = diagbfweights(chanmat,Pt, Pn)
[ ___ ] = diagbfweights(chanmat,Pt, Pn,powdistoption)
```

Description

`[wp,wc] = diagbfweights(chanmat)` returns precoding weights, `wp`, and combining weights, `wc`, for the channel response matrix, `chanmat`. Together, these weights diagonalize the channel into subchannels so that the matrix `wp*chanmat*wc` is diagonal.

`[wp,wc,P] = diagbfweights(chanmat)` also returns the distributed power, `P`, for each element of the transmitting array.

`[wp,wc,P,G] = diagbfweights(chanmat)` also returns the subcarrier gains, `G`.

`[wp,wc,P,G,C] = diagbfweights(chanmat)` also returns the channel capacity sum, `C`.

`[ ___ ] = diagbfweights(chanmat,Pt)` also specifies total transmit power, `Pt`, and returned values any of the previous output argument combinations.

`[ ___ ] = diagbfweights(chanmat,Pt, Pn)` also specifies the noise power per transmitting antenna, `Pn`.

`[ ___ ] = diagbfweights(chanmat,Pt, Pn,powdistoption)` also specifies the noise distribution, `powdistoption`, across all transmitting antennas.

Examples

Compute and Diagonalize Channel Matrix

Compute the channel matrix for a 4-by-4 transmitting URA array and a 5-by-5 receiving URA array. Assume that three scatterers are randomly located within a specified angular range. The element spacing for both arrays is one-half wavelength. The receive array is 500 wavelengths away from the transmitting array along the *x*-axis. Constrain the angular span for the transmitting and receiving arrays. Diagonalize the channel matrix to compute the precoding and combining weights.

Specify the 4-by-4 transmitting array. Element spacing is in units of wavelength.

```
Nt = 4;
sp = 0.5;
```

```
ygridtx = (0:Nt-1)*sp - (Nt-1)/2*sp;
zgridtx = (0:Nt-1)*sp - (Nt-1)/2*sp;
[ytx,ztx] = meshgrid(ygridtx,zgridtx);
txpos = [zeros(1,Nt*Nt);ytx(:).';ztx(:).'];
```

Specify the 5-by-5 receiving array. Element spacing is in units of wavelength.

```
Nr = 5;
sp = 0.5;
ygridrx = (0:Nr-1)*sp - (Nr-1)/2*sp;
zgridrx = (0:Nr-1)*sp - (Nr-1)/2*sp;
[yrx,zrx] = meshgrid(ygridrx,zgridrx);
rxpos = [500*ones(1,Nr*Nr);yrx(:).';zrx(:).'];
```

Set the angular limits for transmitting and receiving.

- The azimuth angle limits for the transmitter are −45° to +45°.
- The azimuth angle limits for the receiver are −75° to +50°.
- The elevation angle limits for the transmitter are −12° to +12°.
- The elevation angle limits for the receiver are −30° to +30°.

```
angrange = [-45 45 -75 50; -12 12 -30 30];
```

Specify three scatterers and create the channel matrix.

```
numscat = 3;
chmat = scatteringchanmtx(txpos,rxpos,numscat,angrange);
```

Diagonalize the channel matrix.

```
[wp,wc] = diagbfweights(chmat);
z = wp*chmat*wc;
```

Show the first four diagonal elements.

```
z(1:4,1:4)
```

```
ans = 4×4 complex

  23.3713 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 - 0.0000i
   0.0000 + 0.0000i  10.7803 + 0.0000i  -0.0000 - 0.0000i  -0.0000 - 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   1.0566 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i  -0.0000 - 0.0000i  -0.0000 + 0.0000i   0.0000 + 0.0000i
```

Distributed Power of Diagonalized Channel Matrix

Compute the channel matrix for a 4-by-4 transmitting URA array and a 5-by-5 receiving URA array. Assume that three scatterers are randomly located within a specified angular range. The element spacings for both arrays is one-half wavelength. The receive array is 500 wavelengths away along the *x*-axis. Diagonalize the channel matrix to compute the precoding and combining weights and the distributed power.

Specify the 4-by-4 transmitting array. Element spacing is in units of wavelength.

```
Nt = 4;
sp = 0.5;
ygridtx = (0:Nt-1)*sp - (Nt-1)/2*sp;
zgridtx = (0:Nt-1)*sp - (Nt-1)/2*sp;
[ytx,ztx] = meshgrid(ygridtx,zgridtx);
txpos = [zeros(1,Nt*Nt);ytx(:).';ztx(:).'];
```

Specify the 5-by-5 receiving array. Element spacing is in units of wavelength.

```
Nr = 5;
sp = 0.5;
ygridrx = (0:Nr-1)*sp - (Nr-1)/2*sp;
zgridrx = (0:Nr-1)*sp - (Nr-1)/2*sp;
[yrx,zrx] = meshgrid(ygridrx,zgridrx);
rxpos = [500*ones(1,Nr*Nr);yrx(:).';zrx(:).'];
```

Set the angular limits for transmitting and receiving.

- The azimuth angle limits for the transmitter are −45° to +45°.
- The azimuth angle limits for the receiver are −75° to +50°.
- The elevation angle limits for the transmitter are −12° to +12°.
- The elevation angle limits for the receiver are −30° to +30°.

```
angrange = [-45 45 -75 50; -12 12 -30 30];
```

Specify three scatterers and create the channel matrix.

```
numscat = 3;
chmat = scatteringchanmtx(txpos,rxpos,numscat,angrange);
```

Diagonalize the channel matrix and return the distributed power.

```
[wp,wc,P] = diagbfweights(chmat);
disp(P.')
```

```
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
    0.0625
```

Subchannel Gains of Diagonalized Channel Matrix

Compute the channel matrix for an 11-element transmitting ULA array and a 7-element receiving ULA array. Assume that there are five randomly located scatterers. The element spacings for both arrays is one-half wavelength. The receive array is 500 wavelengths away from the transmit array along the *x*-axis. Diagonalize the channel matrix to compute the precoding and combining weights, the distributed power, and the subchannel gains.

Specify the 11-element transmitting ULA array. Element spacing is in units of wavelength.

```
Nt = 11;
sp = 0.5;
txpos = (0:Nt-1)*sp - (Nt-1)/2*sp;
```

Specify the 7-element receiving ULA array. Element spacing is in units of wavelength.

```
Nr = 7;
sp = 0.5;
rxpos = (0:Nr-1)*sp - (Nr-1)/2*sp;
numscat = 5;
chmat = scatteringchanmtx(txpos,rxpos,numscat);
```

Diagonalize the channel matrix and return the subchannel gains.

```
[wp,wc,P,G] = diagbfweights(chmat);
disp(G.')
```

```
  221.8345
   56.8443
   47.6711
    0.8143
    0.0000
    0.0000
    0.0000
```

Channel Capacity Sum of Diagonalized Channel Matrix

Compute the channel matrix for an 11-element transmitting ULA array and a 7-element receiving ULA array. Assume that there are five randomly located scatterers. The element spacings for both arrays is one-half wavelength. The receive array is 500 wavelengths away from the transmitting array along the *x*-axis. Create a channel matrix with two subcarriers. Diagonalize the channel matrix to compute the precoding and combining weights, the distributed power, the subchannel gains, and the channel capacity sum.

Specify the 11-element transmitting ULA array. Element spacing is in units of wavelength.

```
Nt = 11;
sp = 0.5;
txpos = (0:Nt-1)*sp - (Nt-1)/2*sp;
```

Specify the 7-element receiving ULA array. Element spacing is in units of wavelength.

```
Nr = 7;
sp = 0.5;
rxpos = (0:Nr-1)*sp - (Nr-1)/2*sp;
numscat = 5;
```

Create two subcarriers.

```
chmat1 = scatteringchanmtx(txpos,rxpos,numscat);
chmat2 = scatteringchanmtx(txpos,rxpos,numscat);
chmat(1,:,:) = chmat1;
chmat(2,:,:) = chmat2;
```

Diagonalize the channel matrix and return the subchannel gains.

```
[wp,wc,P,G,C] = diagbfweights(chmat);
disp(C.')
```

```
    9.5466    9.3605
```

Diagonalize Channel Matrix with Specified Power

Compute the channel matrix for an 11-element transmitting ULA array and a 7-element receiving ULA array. Specify the total transmitted power at 1000. Assume that there are five randomly located scatterers. The element spacings for both arrays is one-half wavelength. The receive array is 500 wavelengths away from the transmitting array along the *x*-axis. Create a channel matrix with two subcarriers. Diagonalize the channel matrix to compute the precoding and combining weights, the distributed power, the subchannel gains, and the channel capacity sum.

Specify the 11-element transmitting ULA array. Element spacing is in units of wavelength.

```
Nt = 11;
sp = 0.5;
txpos = (0:Nt-1)*sp - (Nt-1)/2*sp;
```

Specify the 7-element receiving ULA array. Element spacing is in units of wavelength.

```
Nr = 7;
sp = 0.5;
rxpos = (0:Nr-1)*sp - (Nr-1)/2*sp;
numscat = 5;
```

Create two subcarriers.

```
chmat1 = scatteringchanmtx(txpos,rxpos,numscat);
chmat2 = scatteringchanmtx(txpos,rxpos,numscat);
chmat(1,:,:) = chmat1;
chmat(2,:,:) = chmat2;
```

Diagonalize the channel matrix and return the distributed power for both subcarriers.

```
Pt = 1000.0;
[wp,wc,P,G,C] = diagbfweights(chmat,Pt);
disp(P.')
```

```
   90.9091   90.9091
   90.9091   90.9091
   90.9091   90.9091
   90.9091   90.9091
   90.9091   90.9091
   90.9091   90.9091
   90.9091   90.9091
   90.9091   90.9091
   90.9091   90.9091
```

```
   90.9091   90.9091
   90.9091   90.9091
```

Diagonalize Channel Matrix with Specified Noise Power

Compute the channel matrix for an 11-element transmitting ULA array and a 7-element receiving ULA array. Specify the total transmitted power at 1000 and the transmitting antenna noise power at 100. Assume that there are five randomly located scatterers. The element spacings for both arrays is one-half wavelength. The receive array is 500 wavelengths away from the transmit array along the *x*-axis. Create a channel matrix with two subcarriers. Diagonalize the channel matrix to compute the precoding and combining weights, the distributed power, subchannel gains, and channel capacity sum.

Specify the 11-element transmitting ULA array. Element spacing is in units of wavelength.

```
Nt = 11;
sp = 0.5;
txpos = (0:Nt-1)*sp - (Nt-1)/2*sp;
```

Specify the 7-element receiving ULA array. Element spacing is in units of wavelength.

```
Nr = 7;
sp = 0.5;
rxpos = (0:Nr-1)*sp - (Nr-1)/2*sp;
numscat = 5;
```

Create two subcarriers.

```
chmat1 = scatteringchanmtx(txpos,rxpos,numscat);
chmat2 = scatteringchanmtx(txpos,rxpos,numscat);
chmat(1,:,:) = chmat1;
chmat(2,:,:) = chmat2;
```

Diagonalize the channel matrix and return the gain for both subcarriers.

```
Pt = 1000.0;
Pn = 100.0;
[wp,wc,P,G,C] = diagbfweights(chmat,Pt,Pn);
disp(G.')
```

```
  221.8345  119.7549
   56.8443  115.9814
   47.6711   24.9780
    0.8143    5.1025
    0.0000    0.0059
    0.0000    0.0000
    0.0000    0.0000
```

Diagonalize Channel Matrix Using Waterfill Power Distribution

Compute the channel matrix for an 11-element transmitting ULA array and a 7-element receiving ULA array. Specify the total transmitted power at 1000 and the transmitting antenna noise power at

100. Specify the transmitted power distribution as `'Waterfill'`. Assume that there are five randomly located scatterers. The element spacing for both arrays is one-half wavelength. The receive array is 500 wavelengths away from the transmitting array along the *x*-axis. Create a channel matrix with two subcarriers. Diagonalize the channel matrix to compute the precoding and combining weights, the distributed power, the subchannel gains, and the channel capacity sum.

Specify the 11-element transmitting ULA array. Element spacing is in units of wavelength.

```
Nt = 11;
sp = 0.5;
txpos = (0:Nt-1)*sp - (Nt-1)/2*sp;
```

Specify the 7-element receiving ULA array. Element spacing is in units of wavelength.

```
Nr = 7;
sp = 0.5;
rxpos = (0:Nr-1)*sp - (Nr-1)/2*sp;
numscat = 5;
```

Create two subcarriers.

```
chmat1 = scatteringchanmtx(txpos,rxpos,numscat);
chmat2 = scatteringchanmtx(txpos,rxpos,numscat);
chmat(1,:,:) = chmat1;
chmat(2,:,:) = chmat2;
```

Diagonalize the channel matrix and return the gain for both subcarriers.

```
Pt = 1000.0;
Pn = 100.0;
[wp,wc,P,G,C] = diagbfweights(chmat,Pt,Pn,'Waterfill');
disp(G.')
```

```
  221.8345  119.7549
   56.8443  115.9814
   47.6711   24.9780
    0.8143    5.1025
    0.0000    0.0059
    0.0000    0.0000
    0.0000    0.0000
```

Input Arguments

`chanmat` — Channel response matrix

N_t-by-N_r complex-valued matrix | L-by-N_t-by-N_r complex-valued MATLAB array

Channel response matrix, specified as an N_t-by-N_r complex-valued matrix or an L-by-N_t-by-N_r complex-valued MATLAB array.

- N_t is the number of elements in the transmitting array.
- N_r is the number of elements in the receiving array.
- L is the number of subcarriers.

When `chanmat` is a MATLAB array containing subcarriers, each subcarrier is decomposed independently into subchannels.

Data Types: `double`
Complex Number Support: Yes

`Pt` — Total transmit power
`1` (default) | positive scalar | L-element vector of positive values

Total transmit power, specified as a positive scalar or an L-element vector of positive values. `Pt` has the same units as the total distributed power, `P`.

Data Types: `double`

`Pn` — Noise power
`1` (default) | positive scalar

Noise power in each receiving antenna, specified as a positive scalar. `Pn` has the same units as the total transmit power, `Pt`.

Data Types: `double`

`powdistoption` — Power distribution option
`'Uniform'` (default) | `'Waterfill'`

Power distribution option, specified as `'Uniform'` or `'Waterfill'`. When `powdistoption` is `'Uniform'`, the transmit power is evenly distributed across all N_t channels. If `powdistoption` is `'Waterfill'`, the transmit power is distributed across the N_t channels using a waterfill algorithm.

Data Types: `char`

Output Arguments

`wp` — Precoding weights
N_t-by-N_t complex-valued matrix | L-by-N_t-by-N_t complex-valued MATLAB array

Precoding weights, returned as an N_t-by-N_t complex-valued matrix or an L-by-N_t-by-N_t complex-valued MATLAB array. Units are dimensionless.

Data Types: `double`

`wc` — Combining weights
N_r-by-N_r complex-valued matrix | L-by-N_r-by-N_r complex-valued MATLAB array

Combining weights, returned as an N_r-by-N_r complex-valued matrix or an L-by-N_r-by-N_r complex-valued MATLAB array. Units are dimensionless.

Data Types: `double`

`P` — Distributed power
1-by-N_t real-valued row vector | L-by-N_t real-valued matrix

Distributed power, returned as a vector or matrix.

- When `chanmat` is an N_t-by-N_r real-valued matrix, `P` is a 1-by-N_t real-valued row vector.
- When `chanmat` is an L-by-N_t-by-N_r real-valued MATLAB array, `P` is an L-by-N_t real-valued matrix.

Power units are linear.

Data Types: `double`

G — Subchannel gains
1-by-N_g complex-valued row vector | L-by-N_g complex-valued matrix

Subchannel gains, returned as a vector or matrix.

- When `chanmat` is an N_t-by-N_r complex-valued matrix, `G` is a 1-by-N_g complex-valued row vector.
- When `chanmat` is an L-by-N_t-by-N_r complex-valued MATLAB array, `G` is an L-by-N_g complex-valued matrix.

N_g is the smaller of N_t and N_r.

Gain units are linear.

Data Types: `double`

C — Channel capacity sum for each subcarrier
scalar | L-by-1 vector

Channel capacity sum for each subcarrier, returned as a scalar or vector.

- When `chanmat` is an N_t-by-N_r complex-valued matrix, `C` is a scalar.
- When `chanmat` is an L-by-N_t-by-N_r complex-valued MATLAB array, `C` is an L-by-1 vector.

Capacity units are in bps/Hz.

Data Types: `double`

Version History

Introduced in R2017a

References

[1] Heath, R. Jr. et al. “An Overview of Signal Processing Techniques for Millimeter Wave MIMO Systems”, arXiv.org:1512.03007 [cs.IT], 2015.

[2] Tse, D. and P. Viswanath, *Fundamentals of Wireless Communications*, Cambridge: Cambridge University Press, 2005.

[3] Paulraj, A. *Introduction to Space-Time Wireless Communications*, Cambridge: Cambridge University Press, 2003.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.

See Also

Functions
`scatteringchanmtx` | `waterfill` | `blkdiagbfweights`

Objects
`phased.ScatteringMIMOChannel`

diffbfweights

Differential beamforming weights for ULA and UCA

Syntax

```
wts = diffbfweights(nelem,elemspacing)
wts = diffbfweights(nelem,elemspacing,ArrayGeometry='ULA')
wts = diffbfweights(nelem,rad,ArrayGeometry='UCA')
wts = diffbfweights(nelem,elemspacing,angc,ArrayGeometry=ag)
wts = diffbfweights(nelem,elemspacing,angc,respc,ArrayGeometry=ag)
wts = diffbfweights( ___ ,SteerAngle=angs)
wts = diffbfweights( ___ ,DiffuseNoiseCovariance=dncov)
wts = diffbfweights( ___ ,DiagonalLoading=epsilon)
[wts,pos] = diffbfweights( ___ )
```

Description

`wts = diffbfweights(nelem,elemspacing)` computes differential beamforming weights that steer a uniform linear array (ULA) to its endfire direction. `nelem` is the number of elements in the array and `elemspacing` is the distance between array elements. Differential weights can achieve frequency invariant beam patterns and are often used in microphone arrays.

`wts = diffbfweights(nelem,elemspacing,ArrayGeometry='ULA')` explicitly sets the array to a uniform linear array (ULA).

`wts = diffbfweights(nelem,rad,ArrayGeometry='UCA')` computes the differential beamforming weights for a uniform circular array (UCA). `nelem` is the number of elements in the array and `rad` is the radius ot the array.

`wts = diffbfweights(nelem,elemspacing,angc,ArrayGeometry=ag)` also specifies angles `angc` at which the array response is null. When `ag` equals 'ULA', `elemspacing` refers to the distance between array elements. When `ag` equals 'UCA', `elemspacing` refers to the array radius.

`wts = diffbfweights(nelem,elemspacing,angc,respc,ArrayGeometry=ag)` also specifies the desired responses, `respc`, at the angles specified in `angc`. When `ag` equals 'ULA', `elemspacing` refers to the number of array elements, When `ag` equals 'UCA', `elemspacing` refers to the array radius.

`wts = diffbfweights( ___ ,SteerAngle=angs)` also specifies the array steering angle `angs`.

`wts = diffbfweights( ___ ,DiffuseNoiseCovariance=dncov)` also specifies the diffuse noise covariance matrix `dncov`.

`wts = diffbfweights( ___ ,DiagonalLoading=epsilon)` also specifies the diagonal loading factor `epsilon`.

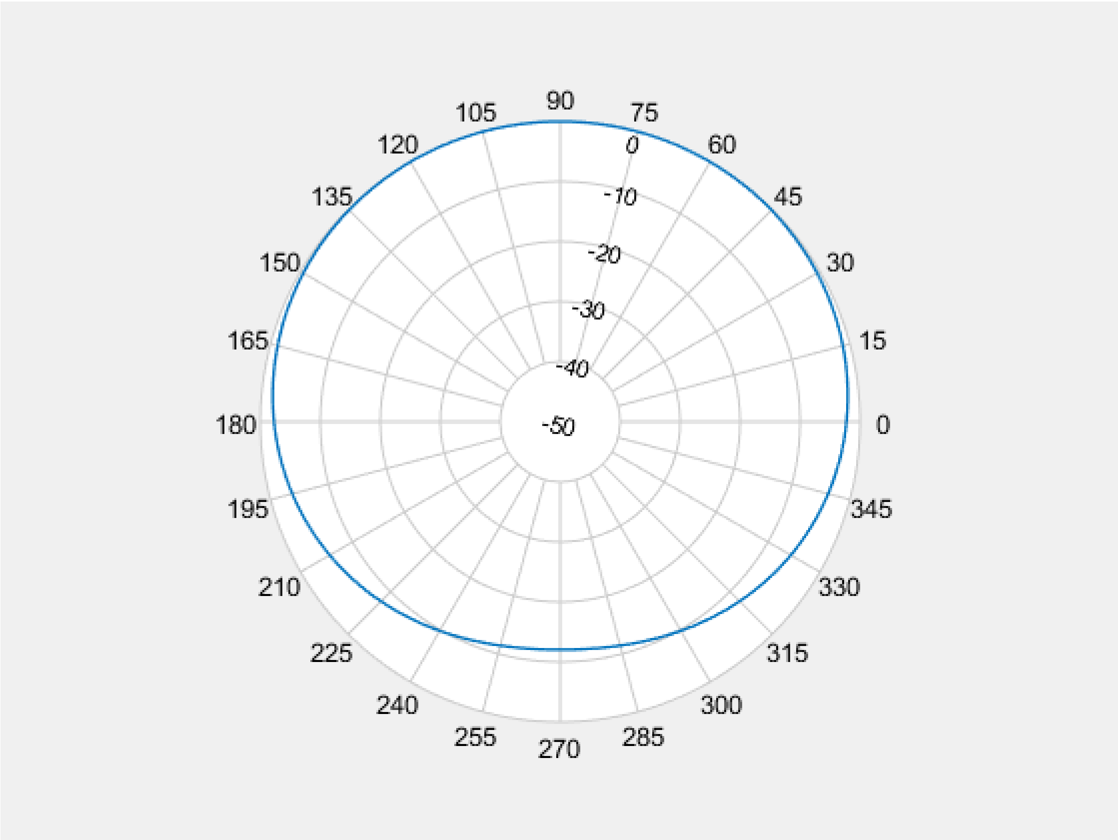
`[wts,pos] = diffbfweights( ___ )` also returns the element positions `pos`.

Examples

Display Pattern of Differentially-Beamformed ULA

Display the pattern of a four-element uniform linear array. The element spacing is 1/10 of the wavelength.

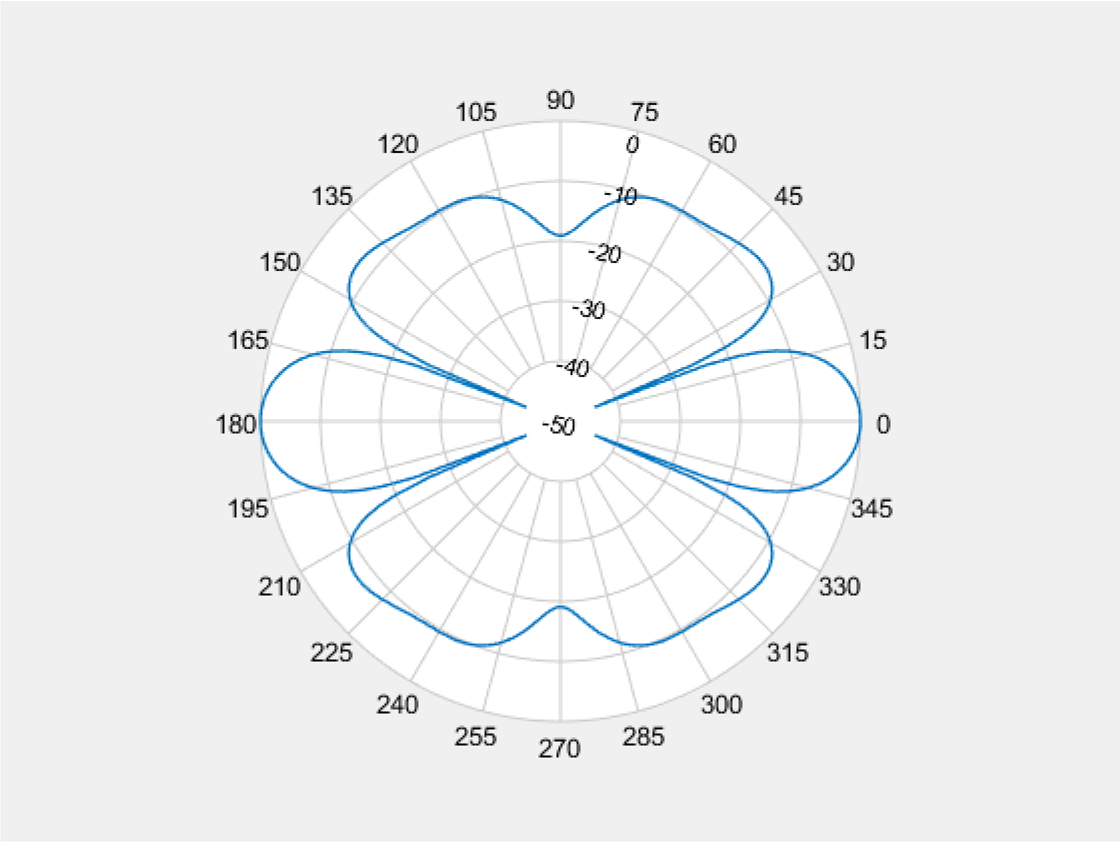
```
N = 4;
angp = -180:180;
[w,pos] = diffbfweights(N,0.1,ArrayGeometry='ULA');
bp = arrayfactor(pos,angp,w);
polarpattern(angp,mag2db(abs(bp)),'NormalizeData',true, ...
    'MagnitudeLimMode','manual','MagnitudeLim',[-50 0])
```



Display Pattern of Differentially-Beamformed UCA

Display the pattern of a six-element uniform circular array. The array radius is one wavelength.

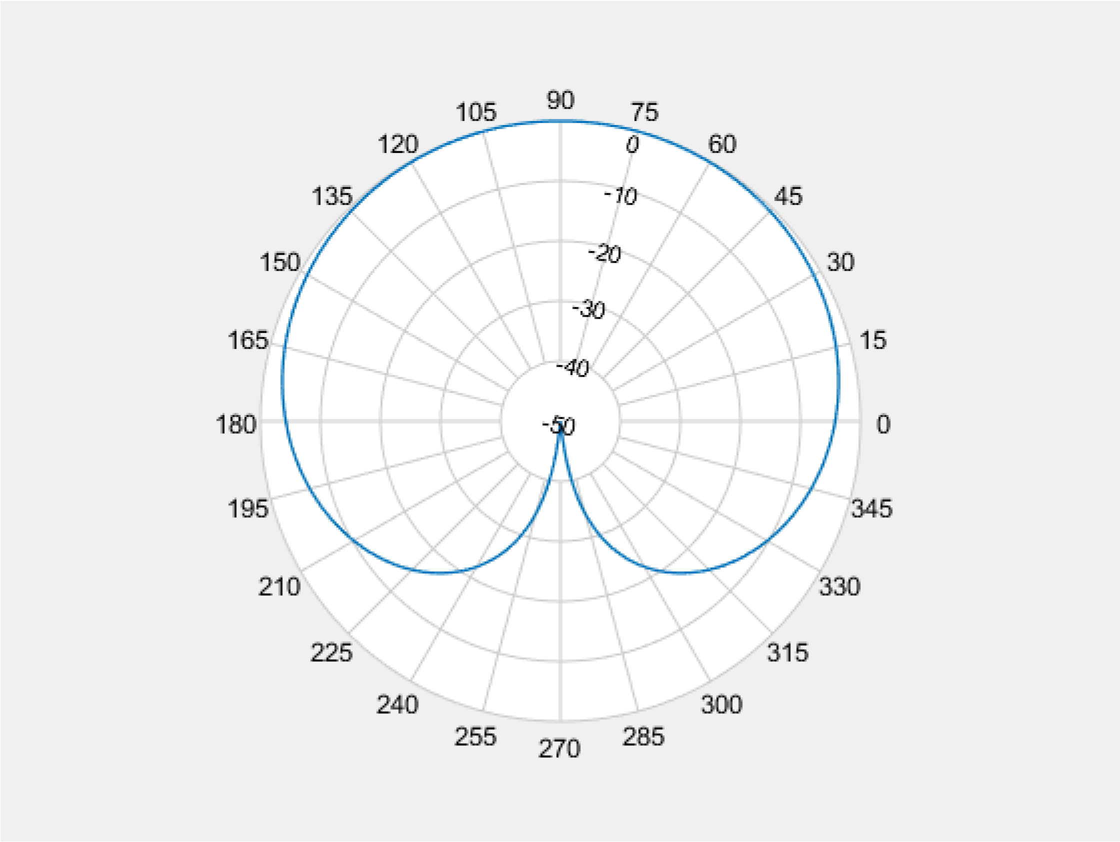
```
N = 6;
angp = -180:180;
[w,pos] = diffbfweights(N,1,ArrayGeometry='UCA');
bp = arrayfactor(pos,angp,w);
polarpattern(angp,mag2db(abs(bp)),'NormalizeData',true, ...
    'MagnitudeLimMode','manual','MagnitudeLim',[-50 0])
```



Cardioid Pattern from Differential Beamforming of ULA

Create a cardioid pattern for a four-element uniform linear array. The main beam points to the endfire direction, 90 degrees azimuth, and the null is in the opposite direction. The element spacing is set to be 1/10 of the wavelength corresponding to the maximum frequency of interest.

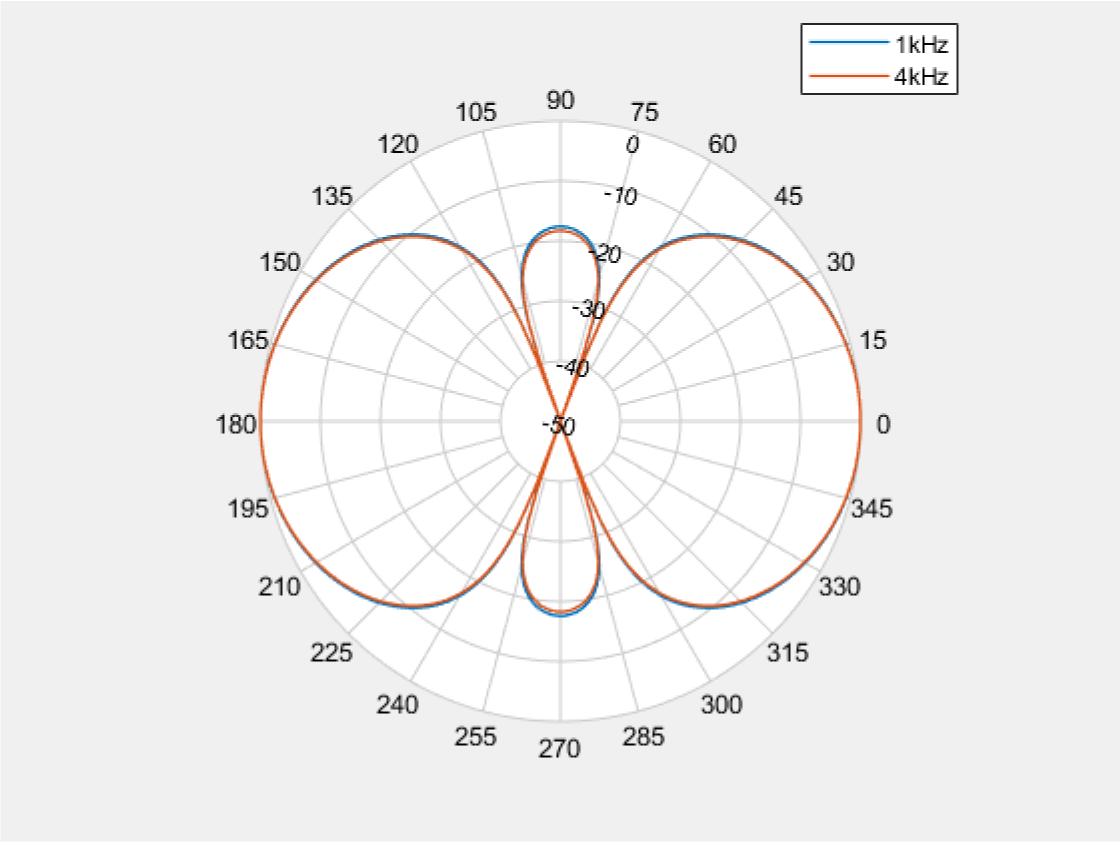
```
N = 4;
angp = -180:180;
[w,pos] = diffbfweights(4,0.1,-90);
bp = arrayfactor(pos,angp,w);
polarpattern(angp,mag2db(abs(bp)),'NormalizeData',true, ...
    'MagnitudeLimMode','manual','MagnitudeLim',[-50 0])
```



Cardioid Pattern of Four-Element ULA

Create a cardioid pattern with a four-element uniform linear array. Plot the pattern for both 1 kHz and 4 kHz. The main beam points to the broadside direction (0 degrees azimuth) and the null is at 70 degrees azimuth. The element spacing is set to be 1/10 of the wavelength corresponding to the maximum frequency of interest,

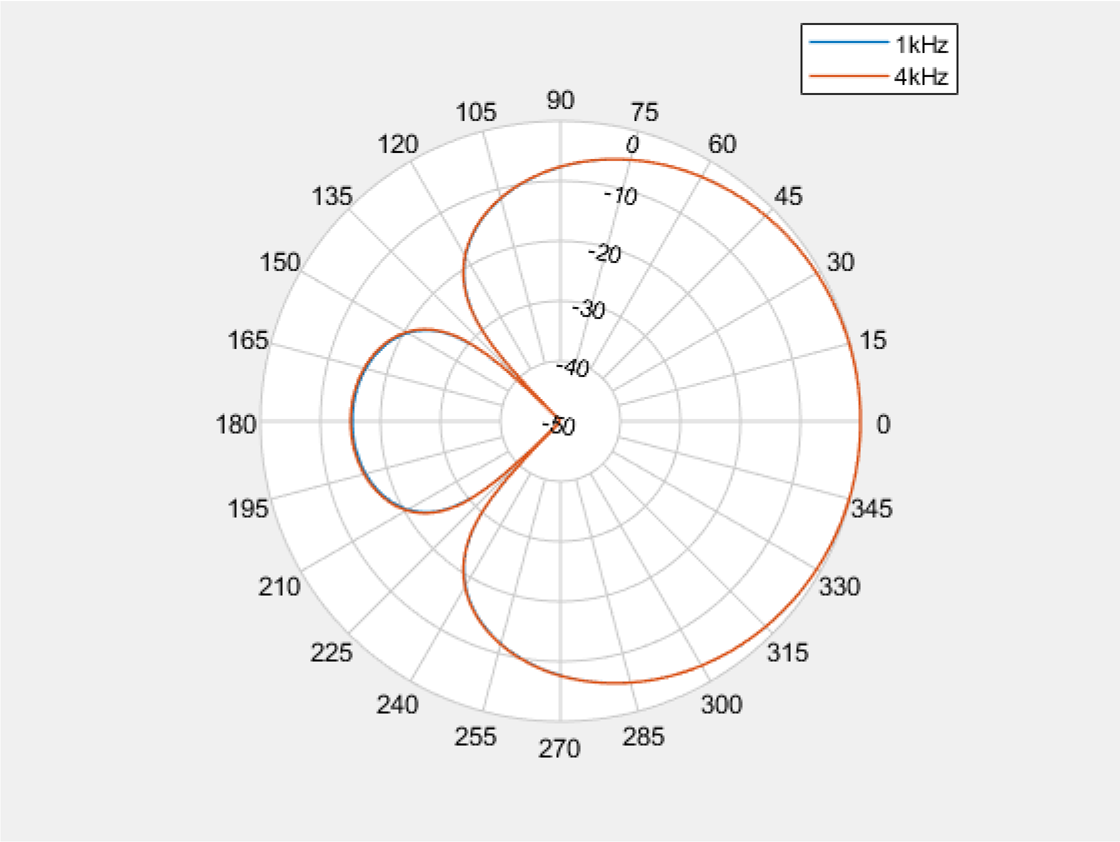
```
N = 4;
c = 343;
fc = [1e3 4e3];
lambda = c./fc;
d = 0.1*lambda(2);
angp = -180:180;
for m = 2:-1:1
    [w,pos] = diffbfweights(N,d/lambda(m),70,SteerAngle=0);
    bp(:,m) = arrayfactor(pos,angp,w);
end
polarpattern(angp,mag2db(abs(bp)),'NormalizeData',true, ...
    'MagnitudeLimMode','manual','MagnitudeLim',[-50 0])
legend('1kHz','4kHz')
```



Super-Cardioid from Three-Element UCA

Create a super-cardioid pattern with a 3-element uniform circular array. Plot the pattern for both 1 kHz and 4 kHz. The main beam points to 0 degrees azimuth and the null is at 135 degrees azimuth. The element spacing is set to be 1/10 of the wavelength corresponding to the maximum frequency of interest.

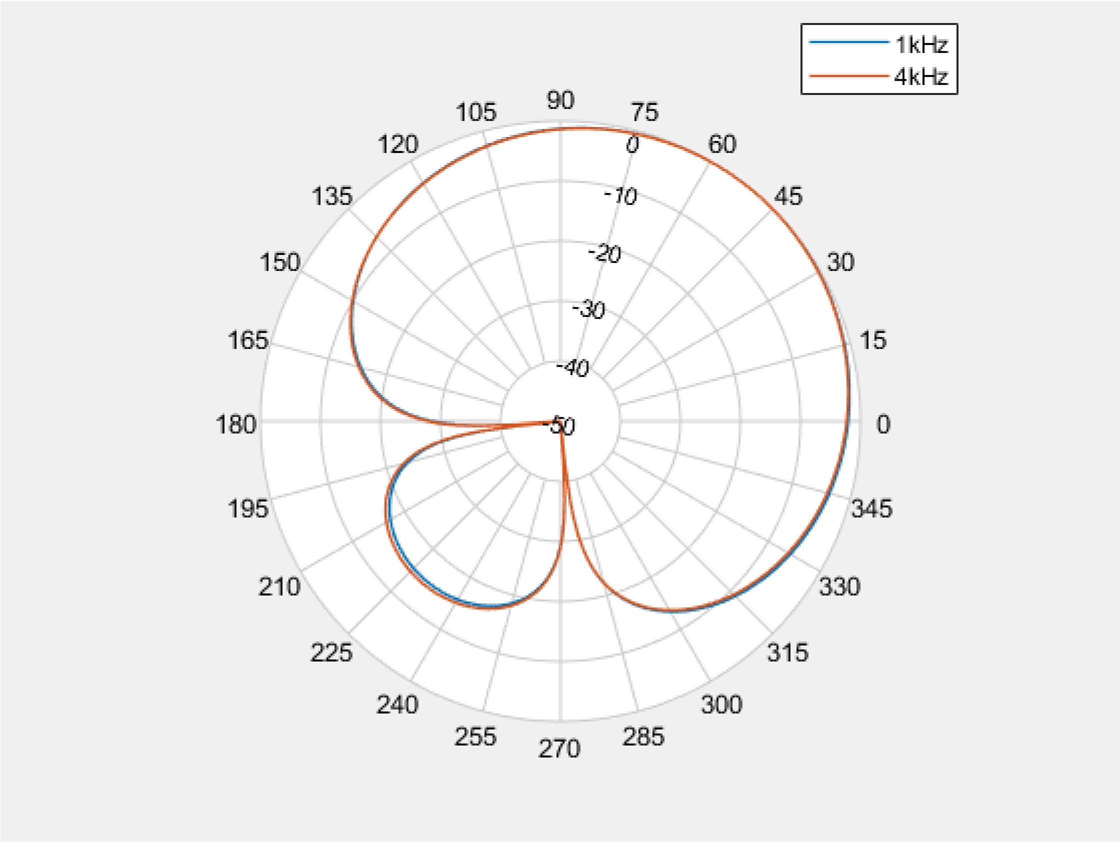
```
N = 3;
c = 343;
fc = [1e3 4e3];
lambda = c./fc;
d = 0.1*lambda(2);
r = N*d/(2*pi);
angp = -180:180;
for m = 2:-1:1
    [w,pos] = diffbfweights(N,r/lambda(m),135,'ArrayGeometry','UCA');
    bp(:,m) = arrayfactor(pos,angp,w);
end
polarpattern(angp,mag2db(abs(bp)),'NormalizeData',true, ...
    'MagnitudeLimMode','manual','MagnitudeLim',[-50 0])
legend('1kHz','4kHz')
```



Super-Cardioid from Four-Element UCA

Create a super-cardioid pattern with a four-element uniform circular array. Plot the pattern for both 1 kHz and 4 kHz. The main beam points to 50 degrees azimuth and the null is at 185 degrees azimuth. The radius is set to be 1/10 of the wavelength corresponding to the maximum frequency of interest.

```
N = 4;
c = 343;
fc = [1e3 4e3];
lambda = c./fc;
r = 0.1*lambda(2);
angp = -180:180;
for m = 2:-1:1
    [w,pos] = diffbfweights(N,r/lambda(m),185, ...
        SteerAngle=50,ArrayGeometry='UCA');
    bp(:,m) = arrayfactor(pos,angp,w);
end
polarpattern(angp,mag2db(abs(bp)),'NormalizeData',true, ...
    'MagnitudeLimMode','manual','MagnitudeLim',[-50 0])
legend('1kHz','4kHz')
```



Input Arguments

nelem — Number of array elements
positive integer

Number of array element, specified as a positive integer.

Data Types: `double`

elemspacing — Element spacing of ULA
positive integer

Element spacing of a ULA array, specified as a positive scalar. Units are in wavelength.

Dependencies

To enable this argument, set the `ArrayGeometry` property to `'ULA'`.

Data Types: `double`

rad — Radius of UCA
positive scalar

Radius of UCA, specified as a positive scalar. Units are in wavelength.

Dependencies

To enable this argument, set the `ArrayGeometry` property to `'UCA'`.

Data Types: `double`

ag — Array geometry
`'ULA'` (default) | `'UCA'`

Array geometry, specified as `'ULA'` or `'UCA'`. `'ULA'` represent a uniform linear array and `'UCA'` represents a uniform circular array. Use this argument to set the value of the `ArrayGeometry` property.

Data Types: `double`

angc — Angular directions of null response
real-valued *P*-element row vector

Angular directions of null responses measured from broadside, specified as a real-valued *P*-element row vector. Angles must lie in the interval [–90,90].

Data Types: `double`

respc — Response at null directions
real-valued *P*-element row vector

Response at the null angles defined by `angc`, specified as a real-valued *P*-element row vector. The number of entries in `respc` must be equal to the number of entries in `angc`.

Data Types: `double`

angs — Steering direction
scalar

Steering vector direction of the array, specified as a scalar in the interval [–90, 90]. The angle is measured from broadside. This argument sets the value of the `SteerAngle` property. Units are in degrees.

Data Types: `double`

dncov — Diffuse covariance noise matrix
N-by-*N* identity matrix (default) | *N*-by-*N* matrix

Diffuse covariance noise matrix, specified as an *N*-by-*N* matrix. Diffuse noise describes the noise presented at different directions. The default value is an identity matrix, indicating the diffuse noise is spatially white. This argument sets the value of the `DiffuseNoiseCovariance` property.

Data Types: `double`

epsilon — Diagonal loading factor
`0` (default) | non-negative scalar

Diagonal loading factor, specified as as a nonnegative scalar. The diagonal loading is used to form a more robust estimate of the covariance. Units are dimensionless. This argument sets the value of the `DiagonalLoading` property.

Data Types: `double`

Output Arguments

`wts` — Array weights
N-element complex-valued vector

Array weights, returned as an *N*-element complex-valued vector. Values are dimensionless.

`pos` — Array element positions
3-by-*N* real-valued matrix

Array element positions, returned as a 3-by-*N* real-valued matrix where *N* is the number of array elements. Each column of `pos` represents the [x;y;z] coordinates of the corresponding element. For a ULA, the returned array is along *y*-axis, located at *(0:N-1)*D*. For a UCA, the returned array is in the *x*-*y*-plane, with its first element on zero degrees azimuth. Units are in wavelengths.

Version History

Introduced in R2022a

References

[1] Benesty, Jacob, et al. *Fundamentals of Differential Beamforming*. Springer Singapore, 2016. DOI.org (Crossref), https://doi.org/10.1007/978-981-10-1046-0.

[2] Benesty, Jacob, and J. Chen. *Study and Design of Differential Microphone Arrays*. Springer, 2013.

[3] Jingdong Chen, Jacob Benesty, and Chao Pan, "On the design and implementation of linear differential microphone arrays", *The Journal of the Acoustical Society of America*, Vol 136, pp 3097, 2014.

[4] Jilu Jin, Gongping Huang, Xuehan Wang, Jingdong Chen, Jacob Benesty, and Israel Cohen, "Steering Study of Linear Differential Microphone Arrays", *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, Vol 29, pp 158, 2020

[5] Jacob Benesty, Jingdong Chen, and Israel Cohen, *Design of Circular Differential Microphone Arrays*, Springer 2015.

[6] Gongping Huang, Israel Cohen, Jingdong Chen, and Jacob Benesty, "Continuously steerable differential beamformers with null constraints for circular microphone arrays", *The Journal of the Acoustical Society of America*, Vol 148, pp 1248, 2020.

See Also

`steervec` | `arrayfactor` | `polarpattern` | `phased.SteeringVector`

dop2speed

Convert Doppler shift to speed

Syntax

```
radvel = dop2speed(dps,lambda)
```

Description

`radvel = dop2speed(dps,lambda)` returns the radial velocity in meters per second. This value corresponds to the one-way Doppler shift `dps` for the wavelength `lambda`.

Examples

Calculate Speed of Car

Calculate the radial velocity of an automobile based on the Doppler shift of a continuous-wave radar. The radar carrier frequency is 24.15 GHz. Assume a doppler shift of 2.880 kHz.

```
f0 = 24.15e9;
lambda = physconst('LightSpeed')/f0;
dopshift = 2.880e3;
radvel = dop2speed(dopshift,lambda)
```

```
radvel = 35.7516
```

The radial velocity is 35.75 meters per second or 80 miles/hour.

Input Arguments

dps — One-way Doppler shift
scalar | vector | matrix

One-way Doppler shift in hertz, specified as a scalar, vector, or matrix.

Data Types: `single` | `double`

`lambda` — Wavelength
positive scalar

Wavelength in meters, specified as a positive scalar.

Data Types: `single` | `double`

Output Arguments

`radvel` — Radial velocity
scalar | vector | matrix

Radial velocity in meters per second, returned as a scalar, vector, or matrix.

More About

Doppler-Radial Velocity Relation

The radial velocity of a source relative to a receiver can be computed from the one-way Doppler shift:

$$V_{s,r} = \Delta f \lambda$$

where $V_{s,r}$ denotes the radial velocity of the source relative to the receiver, Δf is the Doppler shift in hertz, and λ is the carrier frequency wavelength in meters.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

This function supports single and double precision for input data and arguments.

References

[1] Rappaport, T. *Wireless Communications: Principles & Practices*. Upper Saddle River, NJ: Prentice Hall, 1996.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

See Also

`dopsteeringvec` | `speed2dop`

dopsteeringvec

Doppler steering vector

Syntax

```
DSTV = dopsteeringvec(dopplerfreq,numpulses)
DSTV = dopsteeringvec(dopplerfreq,numpulses,PRF)
```

Description

`DSTV = dopsteeringvec(dopplerfreq,numpulses)` returns the temporal (time-domain) Doppler steering vector for a target at a normalized Doppler frequency of `dopplerfreq` in hertz. The pulse repetition frequency is assumed to be 1 Hz.

`DSTV = dopsteeringvec(dopplerfreq,numpulses,PRF)` specifies the pulse repetition frequency, PRF.

Examples

Compute Steering Vector for Doppler Shift

Calculate the steering vector corresponding to a Doppler frequency of 200 Hz. Assume there are 10 pulses and the PRF is 1 kHz.

```
dstv = dopsteeringvec(200,10,1000)
```

```
dstv = 10×1 complex

   1.0000 + 0.0000i
   0.3090 + 0.9511i
  -0.8090 + 0.5878i
  -0.8090 - 0.5878i
   0.3090 - 0.9511i
   1.0000 - 0.0000i
   0.3090 + 0.9511i
  -0.8090 + 0.5878i
  -0.8090 - 0.5878i
   0.3090 - 0.9511i
```

Input Arguments

`dopplerfreq` — Doppler frequencies
scalar | vector

Doppler frequencies in hertz, specified as a scalar or vector. The normalized Doppler frequency is the Doppler frequency divided by the pulse repetition frequency. Every element of `dopplerfreq` must be smaller than or equal to one-half the pulse repetition frequency `PRF`.

Data Types: `single` | `double`

`numpulses` — Number of pulses
positive integer scalar

Number of pulses, specified as a positive integer scalar. The time-domain Doppler steering vector consists of `numpulses` samples taken at intervals of `1/PRF` (slow-time samples).

Data Types: `single` | `double`

`PRF` — Pulse repetition frequency
`1` (default) | positive scalar

Pulse repetition frequency in hertz, specified as a positive scalar. The time-domain Doppler steering vector consists of `numpulses` samples taken at intervals of `1/PRF` (slow-time samples). The normalized Doppler frequency is the Doppler frequency divided by the pulse repetition frequency.

Data Types: `single` | `double`

Output Arguments

`DSTV` — Temporal Doppler steering vector
column vector | matrix

Temporal Doppler steering vector, returned as a column vector or a matrix. `DSTV` has `numpulses` rows and a number of columns equal to the number of elements of `dopplerfreq`.

More About

Temporal Doppler Steering Vector

The temporal (time-domain) steering vector corresponding to a point scatterer is:

$$e^{j2\pi f_d T_p n}$$

where $n = 0,1,2, \ldots, N - 1$ are slow-time samples (one sample from each of N pulses), f_d is the Doppler frequency, and T_p is the pulse repetition interval. The product of the Doppler frequency and the pulse repetition interval is the normalized Doppler frequency.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

This function supports single and double precision for input data and arguments.

This function does not support variable-size inputs.

References

[1] Melvin, W. L. “A STAP Overview,” *IEEE® Aerospace and Electronic Systems Magazine*, Vol. 19, Number 1, 2004, pp. 19–35.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

See Also

`dop2speed` | `speed2dop`

espritdoa

Direction of arrival using TLS ESPRIT

Syntax

```
ang = espritdoa(R,nsig)
ang = espritdoa( ___ ,Name,Value)
```

Description

`ang = espritdoa(R,nsig)` estimates the directions of arrival, `ang`, of a set of plane waves received on a uniform line array (ULA). The estimation employs the TLS ESPRIT, the total least-squares ESPRIT, algorithm. The input arguments are the estimated spatial covariance matrix between sensor elements, `R`, and the number of arriving signals, `nsig`. In this syntax, sensor elements are spaced one-half wavelength apart.

`ang = espritdoa( ___ ,Name,Value)` estimates the directions of arrival with additional options specified by one or more `Name,Value` pair arguments. This syntax can use any of the input arguments in the previous syntax.

Examples

Three Signals Arriving at Half-Wavelength-Spaced ULA

Assume a half-wavelength spaced uniform line array with 10 elements. Three plane waves arrive from the 0°, –25°, and 30° azimuth directions. Elevation angles are 0°. The noise is spatially and temporally white. The SNR for each signal is 5 dB. Find the arrival angles.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25 30];
Nsig = 3;
R = sensorcov(elementPos,angles,db2pow(-5));
doa = espritdoa(R,Nsig)
```

```
doa = 1×3

   30.0000   -0.0000  -25.0000
```

The `espritdoa` function returns the correct angles.

Three Signals Arriving at 0.4-Wavelength-Spaced ULA

Assume a uniform line array with 10 elements. The element spacing is 0.4 wavelength. Three plane waves arrive from the 0°, –25°, and 30° azimuth directions. Elevation angles are 0°. The noise is spatially and temporally white. The SNR for each signal is 5 dB. Find the arrival angles.

```
N = 10;
d = 0.4;
elementPos = (0:N-1)*d;
angles = [0 -25 30];
Nsig = 3;
R = sensorcov(elementPos,angles,db2pow(-5));
doa = espritdoa(R,Nsig,'ElementSpacing',d)
```

```
doa = 1×3

  -25.0000   -0.0000   30.0000
```

`espritdoa` returns the correct angles.

Input Arguments

R — Spatial covariance matrix
complex-valued positive-definite *N*-by-*N* matrix.

Spatial covariance matrix, specified as a complex-valued, positive-definite, *N*-by-*N* matrix. In this matrix, *N* represents the number of elements in the ULA array. If `R` is not Hermitian, a Hermitian matrix is formed by averaging the matrix and its conjugate transpose, `(R+R')/2`.

Example: [4.3162, –0.2777 – 0.2337i; –0.2777 + 0.2337i , 4.3162]

Data Types: `double`
Complex Number Support: Yes

`nsig` — Number of arriving signals
positive integer

Number of arriving signals, specified as a positive integer.

Example: 3

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: ‘ElementSpacing’, 0.45

`ElementSpacing` — ULA element spacing
0.5 (default) | real-valued positive scalar

ULA element spacing, specified as a real-valued, positive scalar. Position units are measured in terms of signal wavelength.

Example: 0.4

Data Types: `double`

`RowWeighting` — Row weights
1 (default) | real-valued positive scalar

Row weights specified as a real-valued positive scalar. These weights are applied to the selection matrices which determine the ESPRIT subarrays. A larger value is generally better but the value must be less than or equal to $(N_s–1)/2$, where N_s is the number of subarray elements. The number of subarray elements is $N_s = N–1$. The value of N is the number of ULA elements, as specified by the dimensions of the spatial covariance matrix, `R`. A detailed discussion of selection matrices and row weighting can be found in Van Trees [1], p. 1178.

Example: 5

Data Types: `double`

Output Arguments

ang — Directions of arrival angles
real-valued 1-by-M row vector

Directions of arrival angle returned as a real-valued, 1-by-M vector. The dimension M is the number of arriving signals specified in the argument, `nsig`. This angle is the broadside angle. Angle units are degrees and angle values lie between –90° and 90°.

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`aictest` | `mdltest` | `rootmusicdoa` | `spsmooth` | `phased.ESPRITEstimator`

fogpl

RF signal attenuation due to fog and clouds

Syntax

`L = fogpl(R,freq,T,den)`

Description

`L = fogpl(R,freq,T,den)` returns attenuation, `L`, when signals propagate in fog or clouds. `R` represents the signal path length. `freq` represents the signal carrier frequency, `T` is the ambient temperature, and `den` specifies the liquid water density in the fog or cloud.

The `fogpl` function applies the International Telecommunication Union (ITU) cloud and fog attenuation model to calculate path loss of signals propagating through clouds and fog. See [1]. Fog and clouds are the same atmospheric phenomenon, differing only by height above ground. Both environments are parametrized by their liquid water density. Other model parameters include signal frequency and temperature. This function applies to cases when the signal path is contained entirely in a uniform fog or cloud environment. The liquid water density does not vary along the signal path. The attenuation model applies only for frequencies at 10–1000 GHz.

Examples

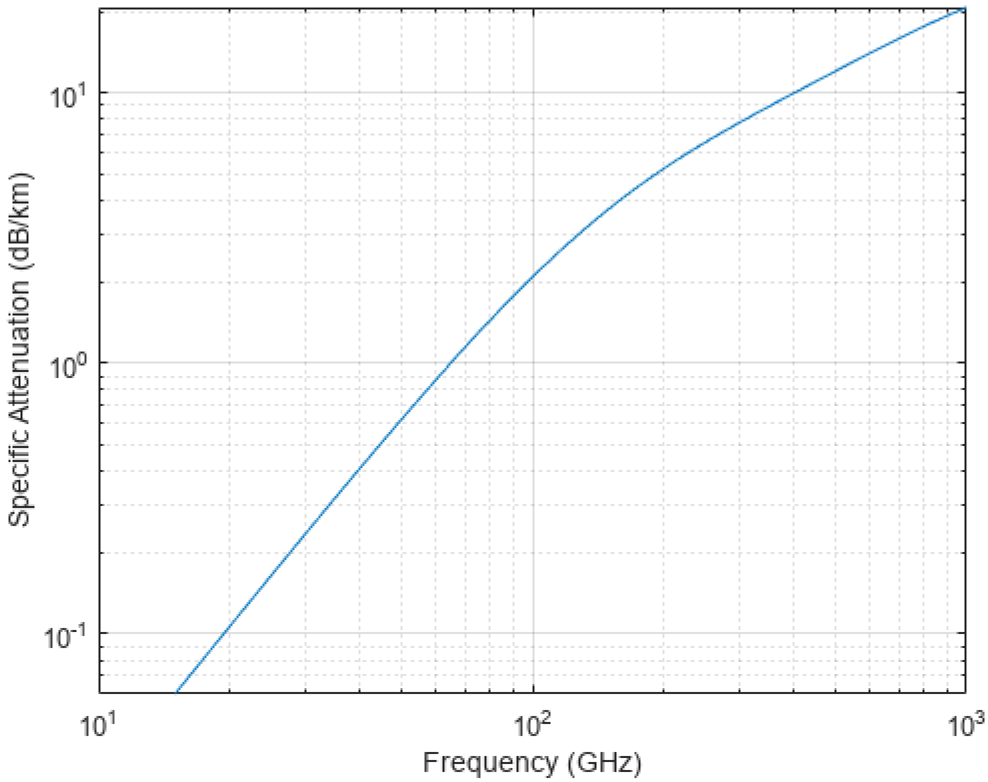
Attenuation in Cumulus Clouds

Compute the attenuation of signals propagating through a cloud that is 1 km long at 1000 meters altitude. Compute the attenuation for frequencies from 15 to 1000 GHz. A typical value for the cloud liquid water density is 0.5 g/m^3. Assume the atmospheric temperature at 1000 meters is 20˚C.

```
R = 1000.0;
freq = [15:5:1000]*1e9;
T = 20.0;
lwd = 0.5;
L = fogpl(R,freq,T,lwd);
```

Plot the specific attenuation as a function of frequency. Specific attenuation is the attenuation or loss per kilometer.

```
loglog(freq/1e9,L)
grid
xlabel('Frequency (GHz)')
ylabel('Specific Attenuation (dB/km)')
```



Input Arguments

R — Signal path length

positive real-valued scalar | M-by-1 nonnegative real-valued vector | 1-by-M nonnegative real-valued vector

Signal path length, specified as a scalar or as an M-by-1 or 1-by-M vector of nonnegative real-values. Total attenuation is the specific attenuation multiplied by the path length. Units are meters.

Example: `[1300.0,1400.0]`

`freq` — Signal frequency

positive real-valued scalar | N-by-1 nonnegative real-valued column vector | 1-by-N nonnegative real-valued row vector

Signal frequency, specified as a positive real-valued scalar or as an N-by-1 nonnegative real-valued vector or 1-by-N nonnegative real-valued vector. Frequencies must lie in the range 10–1000 GHz. Units are in Hz.

Example: `[14.0e9,15.0e9]`

T — Ambient temperature

real-valued scalar

Ambient temperature in fog or cloud, specified as a real-valued scalar. Units are in degrees Celsius.

Example: `-10.0`

den — Liquid water density
nonnegative real-valued scalar

Liquid water density, specified as a nonnegative real-valued scalar. Units are g/m^3. Typical values for liquid water density in fog range from approximately 0.05 g/m^3 for medium fog to approximately 0.5 g/m^3 for thick fog. For medium fog, visibility is about 300 meters. For heavy fog, visibility is about 50 meters. Cumulus cloud liquid water density is typically 0.5 g/m^3.

Example: `0.01`

Output Arguments

L — Signal attenuation
real-valued *M*-by-*N* matrix

Signal attenuation, returned as a real-valued *M*-by-*N* matrix. Each matrix row represents a different path where *M* is the number of paths. Each column represents a different frequency where *N* is the number of frequencies. Units are in dB.

More About

Fog and Cloud Attenuation Model

This model calculates the attenuation of signals that propagate through fog or clouds.

Fog and cloud attenuation are the same atmospheric phenomenon. The ITU model, *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog* is used. The model computes the specific attenuation (attenuation per kilometer), of a signal as a function of liquid water density, signal frequency, and temperature. The model applies to polarized and nonpolarized fields. The formula for specific attenuation at each frequency is

$$\gamma_c = K_l(f)M,$$

where M is the liquid water density in gm/m^3. The quantity $K_l(f)$ is the specific attenuation coefficient and depends on frequency. The cloud and fog attenuation model is valid for frequencies 10–1000 GHz. Units for the specific attenuation coefficient are (dB/km)/(g/m^3).

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length R. Total attenuation is $L_c = R\gamma_c$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply narrowband attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Version History

Introduced in R2016a

References

[1] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog*. 2013.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`fspl` | `rainpl` | `gaspl` | `LOSChannel` | `WidebandLOSChannel`

fspl

Free space path loss

Syntax

`L = fspl(R,lambda)`

Description

`L = fspl(R,lambda)` returns the free space path loss in decibels for a waveform with wavelength `lambda` propagated over a distance of `R` meters. The minimum value of `L` is zero, indicating no path loss.

Examples

Calculate Free-Space Path Loss

Calculate the free-space path loss (in dB) of a 10 GHz radar signal over a distance of 10 km.

```
fc = 10.0e9;
lambda = physconst('LightSpeed')/fc;
R = 10e3;
L = fspl(R,lambda)
```

```
L = 132.4478
```

Input Arguments

R — Propagation distance of signal
real-valued 1-by-*M* or *M*-by-1 vector

Units are in meters.

lambda — Speed of propagation divided by the signal frequency
real-valued 1-by-*N* or *N*-by-1 vector

Wavelength units are meters.

Output Arguments

L — Path loss in decibels
M-by-*N* non-negative matrix. A value of zero signifies no path loss.

When `lambda` is a scalar, `L` has the same dimensions as `R`.

More About

Free Space Path Loss

The free-space path loss, L, in decibels is:

$$L = 20\log_{10}\left(\frac{4\pi R}{\lambda}\right)$$

This formula assumes that the target is in the far-field of the transmitting element or array. In the near-field, the free-space path loss formula is not valid and can result in a loss smaller than 0 dB, equivalent to a signal gain. For this reason, the loss is set to 0 dB for range values $R \leq \lambda/4\pi$.

Version History

Introduced in R2011a

References

[1] Proakis, J. *Digital Communications*. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`gaspl` | `fogpl` | `rainpl` | `phased.FreeSpace`

gain2aperture

Convert gain to effective aperture

Syntax

`A = gain2aperture(GdB,lambda)`

Description

`A = gain2aperture(GdB,lambda)` returns the effective aperture of an antenna corresponding to an antenna gain of `GdB` for an incident electromagnetic wave with wavelength `lambda`.

Examples

Compute Effective Aperture

An antenna has a gain of 3 dB. Calculate the antenna's effective aperture when used to capture an electromagnetic wave with a wavelength of 10 cm.

```
a = gain2aperture(3,0.1)
```

```
a = 0.0016
```

Input Arguments

`GdB` — Antenna gains
scalar | *N*-element real-valued vector

Antenna gains, specified as a scalar or as an *N*-element real-valued vector. If `GdB` is a vector, each element of `GdB` corresponds to the effective aperture of the same element in the output argument `A`. See "Gain and Effective Aperture" on page 2-20 for a discussion of aperture and gain. Units are in dBi.

Data Types: `double`

`lambda` — Wavelength of the incident electromagnetic wave
positive scalar

Wavelength of the incident electromagnetic wave, specified as a positive scalar. The wavelength of an electromagnetic wave is the ratio of the wave propagation speed to the frequency. Units are in meters.

Data Types: `double`

Output Arguments

`A` — Antenna effective aperture
positive scalar | *N*-element vector of positive values

Antenna effective aperture, returned as a positive scalar or as an N-element vector of positive values. The elements of `A` represent the effective apertures for the corresponding elements of `GdB`. The size of `A` equals the size of `GdB`.

Data Types: `double`

More About

Gain and Effective Aperture

The effective aperture describes how much energy is captured by an antenna from an incident electromagnetic plane wave. The effective area of the antenna and is not the same as the actual physical area. The array gain of an antenna G is related to its effective aperture A_e by:

$$G = \frac{4\pi}{\lambda^2}A_e$$

where λ is the wavelength of the incident electromagnetic wave. For a fixed wavelength, the antenna gain is proportional to the effective aperture. For a fixed effective aperture, the antenna gain is inversely proportional to the square of the wavelength.

The gain expressed in dBi (GdB) is

$$GdB = 10\log_{10}G = 10\log_{10}(\frac{4\pi A_g}{\lambda^2}) .$$

The effective antenna aperture can be derived from the gain in dB using

$$A_e = 10^{GdB/10}\frac{\lambda^2}{4\pi} .$$

Version History

Introduced in R2011a

References

[1] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

[2] Richards, M. *Fundamentals of Radar Signal Processing*, New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`aperture2gain`

gaspl

RF signal attenuation due to atmospheric gases

Syntax

```
L = gaspl(range,freq,T,P,den)
```

Description

`L = gaspl(range,freq,T,P,den)` returns the attenuation, `L`, of signals propagating through the atmosphere.

- `range` represents the signal path length.
- `freq` represents the signal carrier frequency.
- `T` represents the ambient temperature.
- `P` represents the atmospheric pressure.
- `den` represents the atmospheric water vapor density.

The `gaspl` function applies the International Telecommunication Union (ITU) atmospheric gas attenuation model [1] to calculate path loss for signals primarily due to oxygen and water vapor. The model computes attenuation as a function of ambient temperature, pressure, water vapor density, and signal frequency.

The function requires that the signal path is contained entirely in a homogeneous environment – temperature `T`, atmospheric pressure `P`, and water vapor density `den` do not vary along the signal path. You can account for the variation of atmospheric parameters with height using the `tropopl` and `atmositu` functions in the Radar Toolbox.

The attenuation model applies only for frequencies at 1–1000 GHz.

Examples

Atmospheric Gas Attenuation Spectrum

Compute the attenuation spectrum from 1 to 1000 GHz for an atmospheric pressure of 101.300 kPa and a temperature of 15°C. Plot the spectrum for a water vapor density of 7.5 g/m^3 and then plot the spectrum for dry air (zero water vapor density).

Set the attenuation frequencies.

```
freq = [1:1000]*1e9;
```

Assume a 1 km path distance.

```
R = 1000.0;
```

Compute the attenuation for air containing water vapor.

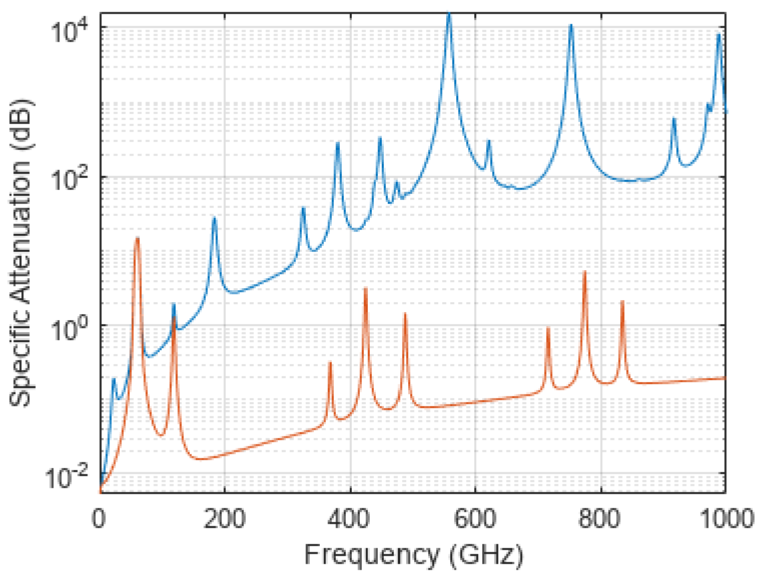
```
T = 15;
P = 101300.0;
W = 7.5;
L = gaspl(R,freq,T,P,W);
```

Compute the attenuation for dry air.

```
L0 = gaspl(R,freq,T,P,0.0);
```

Plot the attenuations.

```
semilogy(freq/1e9,L)
hold on
semilogy(freq/1e9,L0)
grid
xlabel('Frequency (GHz)')
ylabel('Specific Attenuation (dB)')
hold off
```



Plot Attenuation Due to Atmospheric Gases and Free Space

First, plot the specific attenuation of atmospheric gases for frequencies from 1 GHz to 1000 GHz. Assume a sea-level dry air pressure of 101.325e5 kPa and a water vapor density of 7.5 g/m^3. The air temperature is 20˚C. Specific attenuation is defined as dB loss per kilometer. Then, plot the actual attenuation at 10 GHz for a span of ranges.

Plot Specific Atmospheric Gas Attenuation

Set the atmosphere temperature, pressure, water vapor density.

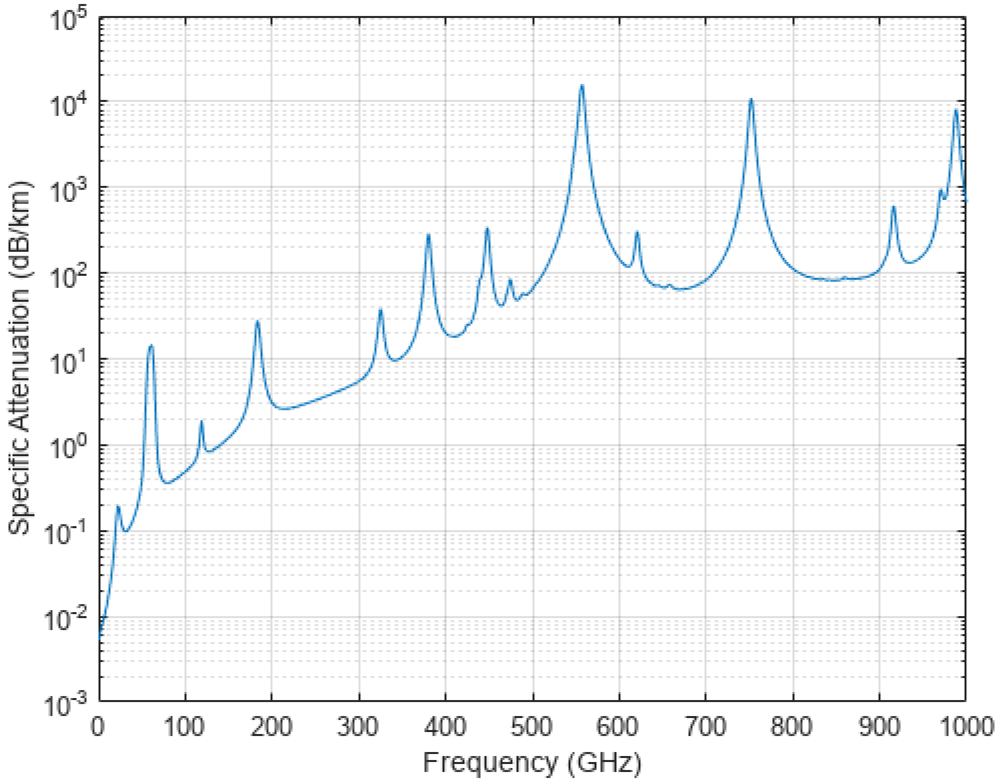
```
T = 20.0;
Patm = 101.325e3;
rho_wv = 7.5;
```

Set the propagation distance, speed of light, and frequencies.

```
km = 1000.0;
c = physconst('LightSpeed');
freqs = [1:1000]*1e9;
```

Compute and plot the atmospheric gas loss.

```
loss = gaspl(km,freqs,T,Patm,rho_wv);
semilogy(freqs/1e9,loss)
grid on
xlabel('Frequency (GHz)')
ylabel('Specific Attenuation (dB/km)')
```

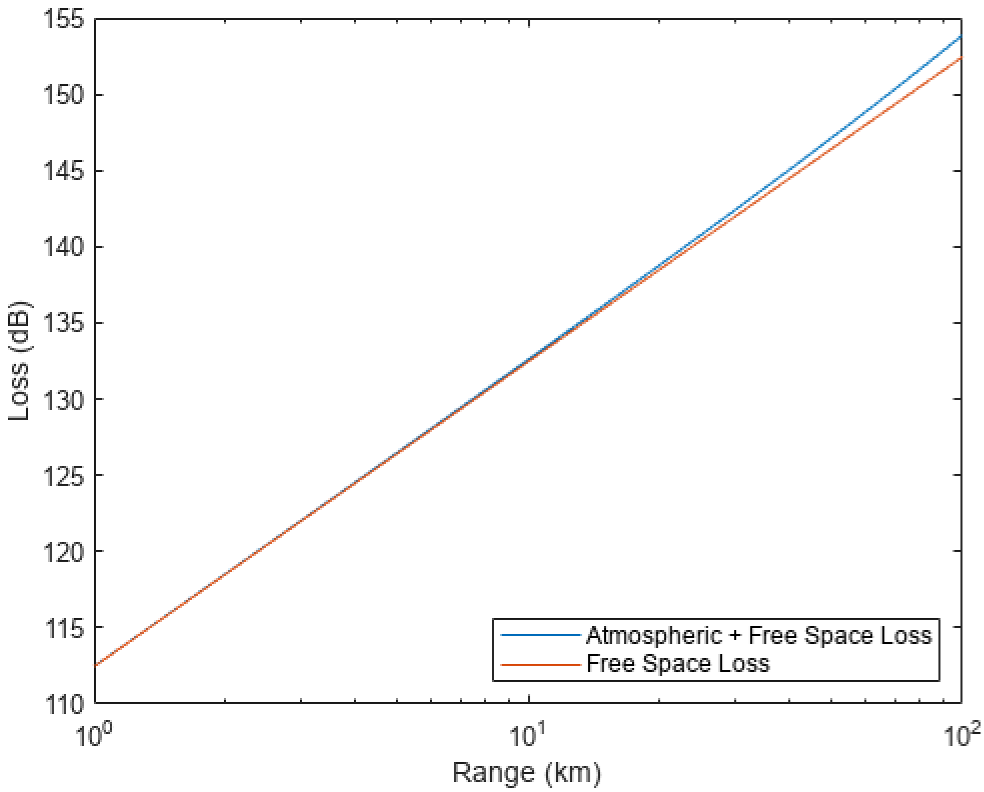


Plot Actual Atmospheric and Free Space Attenuation

Compute both free space loss and atmospheric gas loss at 10 GHz for ranges from 1 to 100 km. The frequency corresponds to an *X*-band radar. Then, plot the free space loss and the total (atmospheric + free space) loss.

```
ranges = [1:100]*1000;
freq_xband = 10e9;
loss_gas = gaspl(ranges,freq_xband,T,Patm,rho_wv);
lambda = c/freq_xband;
loss_fsp = fspl(ranges,lambda);
semilogx(ranges/1000,loss_gas + loss_fsp.',ranges/1000,loss_fsp)
legend('Atmospheric + Free Space Loss','Free Space Loss','Location','SouthEast')
```

```
xlabel('Range (km)')
ylabel('Loss (dB)')
```



Input Arguments

range — Signal path length

nonnegative real-valued scalar | *M*-by-1 nonnegative real-valued column vector | 1-by-*M* nonnegative real-valued row vector

Signal path length used to compute attenuation, specified as a nonnegative real-valued scalar or vector. You can specify multiple path lengths simultaneously. Units are in meters.

Example: `[13000.0,14000.0]`

freq — Signal frequency

positive real-valued scalar | *N*-by-1 nonnegative real-valued column vector | 1-by-*N* nonnegative real-valued row vector

Signal frequency, specified as a positive real-valued scalar, or as an *N*-by-1 nonnegative real-valued vector or 1-by-*N* nonnegative real-valued vector. You can specify multiple frequencies simultaneously. Frequencies must lie in the range 1–1000 GHz. Units are in hertz.

Example: `[1.4e9,2.0e9]`

T — Ambient temperature

real-valued scalar

Ambient temperature, specified as a real-valued scalar. Units are in degrees Celsius.

Example: `-10.0`

P — Dry air pressure
positive real-valued scalar

Dry air pressure, specified as a positive real-valued scalar. Units are in Pa. One standard atmosphere at sea level is 101325 Pa.

Example: `101300.0`

den — Water vapor density
nonnegative real-valued scalar

Water vapor density or absolute humidity, specified as a nonnegative real-valued scalar. Units are g/m^3. The maximum water vapor density of air at 30° C is approximately 30.0 g/m^3. The maximum water vapor density of air at 0°C is approximately 5.0 g/m^3.

Example: `4.0`

Output Arguments

L — Signal attenuation
real-valued *M*-by-*N* matrix

Signal attenuation, returned as a real-valued *M*-by-*N* matrix. Each matrix row represents a different path where *M* is the number of paths. Each column represents a different frequency where *N* is the number of frequencies. Units are in dB.

More About

Atmospheric Gas Attenuation Model

This model calculates the attenuation of signals that propagate through atmospheric gases.

Electromagnetic signals attenuate when they propagate through the atmosphere. This effect is due primarily to the absorption resonance lines of oxygen and water vapor, with smaller contributions coming from nitrogen gas. The model also includes a continuous absorption spectrum below 10 GHz. The ITU model *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* is used. The model computes the specific attenuation (attenuation per kilometer) as a function of temperature, pressure, water vapor density, and signal frequency. The atmospheric gas model is valid for frequencies from 1–1000 GHz and applies to polarized and nonpolarized fields.

The formula for specific attenuation at each frequency is

$$\gamma = \gamma_o(f) + \gamma_w(f) = 0.1820fN''(f).$$

The quantity *N"()* is the imaginary part of the complex atmospheric refractivity and consists of a spectral line component and a continuous component:

$$N''(f) = \sum_i S_i F_i + N''_D(f)$$

The spectral component consists of a sum of discrete spectrum terms composed of a localized frequency bandwidth function, $F(f)_i$, multiplied by a spectral line strength, S_i. For atmospheric oxygen, each spectral line strength is

$$S_i = a_1 \times 10^{-7}\left(\frac{300}{T}\right)^3 \exp\left[a_2(1 - \left(\frac{300}{T}\right)\right]P \,.$$

For atmospheric water vapor, each spectral line strength is

$$S_i = b_1 \times 10^{-1}\left(\frac{300}{T}\right)^{3.5} \exp\left[b_2(1 - \left(\frac{300}{T}\right)\right]W \,.$$

P is the dry air pressure, W is the water vapor partial pressure, and T is the ambient temperature. Pressure units are in hectoPascals (hPa) and temperature is in degrees Kelvin. The water vapor partial pressure, W, is related to the water vapor density, ρ, by

$$W = \frac{\rho T}{216.7} \,.$$

The total atmospheric pressure is $P + W$.

For each oxygen line, S_i depends on two parameters, a_1 and a_2. Similarly, each water vapor line depends on two parameters, b_1 and b_2. The ITU documentation cited at the end of this section contains tabulations of these parameters as functions of frequency.

The localized frequency bandwidth functions $F_i(f)$ are complicated functions of frequency described in the ITU references cited below. The functions depend on empirical model parameters that are also tabulated in the reference.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length, R. Then, the total attenuation is $L_g= R(\gamma_o + \gamma_w)$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Version History

Introduced in R2016a

References

[1] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* 2013.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`fspl` | `fogpl` | `rainpl` | `rainpl` | `phased.LOSChannel` | `phased.WidebandLOSChannel`

gccphat

Generalized cross-correlation

Syntax

```
tau = gccphat(sig,refsig)
tau = gccphat(sig,refsig,fs)
[tau,R,lag] = gccphat( ___ )
```

```
[ ___ ] = gccphat(sig)
[ ___ ] = gccphat(sig,fs)
```

Description

`tau = gccphat(sig,refsig)` computes the time delay, `tau`, between the signal, `sig`, and a reference signal, `refsig`. Both `sig` and `refsig` can have multiple channels. The function assumes that the signal and reference signal come from a single source. To estimate the delay, `gccphat` finds the location of the peak of the cross-correlation between `sig` and `refsig`. The cross-correlation is computed using the generalized cross-correlation phase transform (GCC-PHAT) algorithm. Time delays are multiples of the sample interval corresponding to the default sampling frequency of one hertz.

`tau = gccphat(sig,refsig,fs)`, specifies the sampling frequency of the signal. Time delays are multiples of the sample interval corresponding to the sampling frequency. All input signals should have the same sample rate.

`[tau,R,lag] = gccphat( ___ )` returns, in addition, the cross-correlation values and correlation time lags, using any of the arguments from previous syntaxes. The lags are multiples of the sampling interval. The number of cross-correlation channels equals the number of channels in `sig`.

`[ ___ ] = gccphat(sig)` or `[ ___ ] = gccphat(sig,fs)` returns the estimated delays and cross correlations between all pairs of channels in `sig`. If `sig` has M columns, the resulting `tau` and `R` have M^2 columns. In these syntaxes, no reference signal input is used. The first M columns of `tau` and `R` contain the delays and cross correlations that use the first channel as the reference. The second M columns contain the delays and cross-correlations that use the second channel as the reference, and so on.

Examples

Cross-Correlation Between Two Signals and Reference Signal

Load a gong sound signal. First, use the gong signal as a reference signal. Then, duplicate the signal twice, introducing time delays of 5 and 25 seconds. Leave the sampling rate to its default of one hertz. Use `gccphat` to estimate the time delays between the delayed signals and the reference signal.

```
load gong;
refsig = y;
delay1 = 5;
```

```
delay2 = 25;
sig1 = delayseq(refsig,delay1);
sig2 = delayseq(refsig,delay2);
tau_est = gccphat([sig1,sig2],refsig)
```

```
tau_est = 1×2

     5    25
```

Cross-Correlation Between Signal and Reference Signal

Load a gong sound signal. Use the gong signal as a reference signal. Then, duplicate the signal, introducing a time delays of 5 milliseconds. Use the sampling rate of 8192 Hz. Use `gccphat` to estimate the time delay between the delayed signal and the reference signal.

```
load gong;
delay = 0.005;
refsig = y;
sig = delayseq(refsig,delay,Fs);
tau_est = gccphat(sig,refsig,Fs)
```

```
tau_est = 0.0050
```

Plot Cross-Correlation of Three Signals with Reference Signal

Load a musical sound signal with a sample rate is 8192 hertz. Then, duplicate the signal three times and introduce time delays between the signals. Estimate the time delays between the delayed signals and the reference signals. Plot the correlation values.

```
load handel;
dt = 1/Fs;
refsig = y;
```

Create three delayed versions of the signal.

```
delay1 = -5.2*dt;
delay2 = 10.3*dt;
delay3 = 7*dt;
sig1 = delayseq(refsig,delay1,Fs);
sig2 = delayseq(refsig,delay2,Fs);
sig3 = delayseq(refsig,delay3,Fs);
```

Cross-correlate the delayed signals with the reference signal.

```
[tau_est,R,lags] = gccphat([sig1,sig2,sig3],refsig,Fs);
```

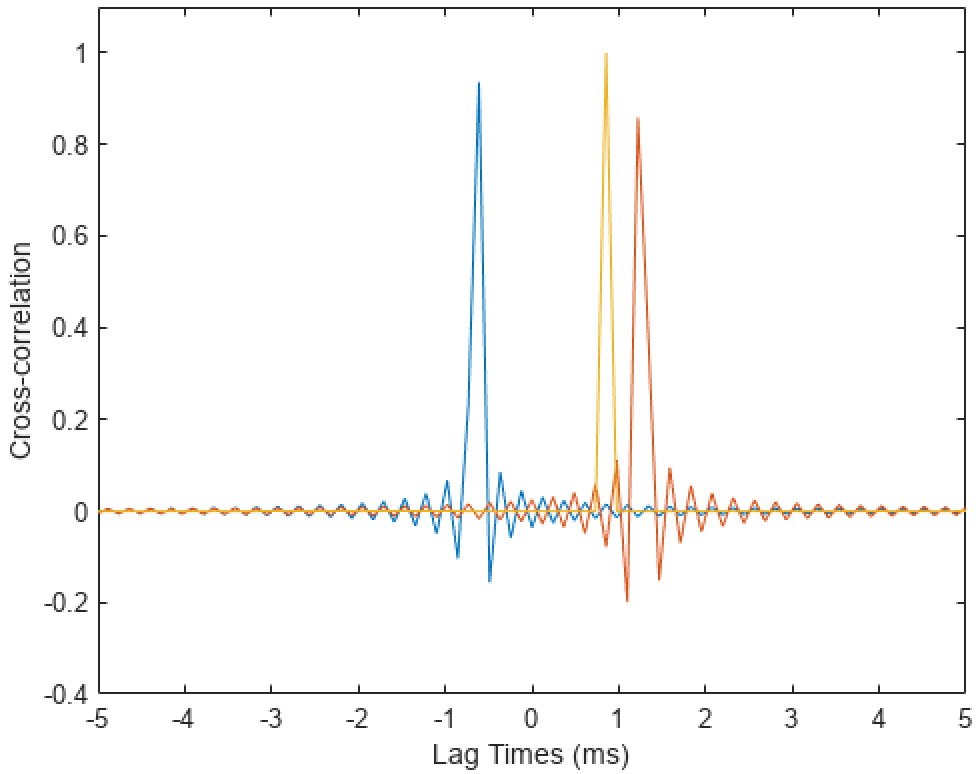
The `gccphat` functions estimates the delay to the nearest sample interval.

```
disp(tau_est*Fs)
```

```
    -5    10     7
```

Plot the correlation functions.

```
plot(1000*lags,real(R(:,1)))
xlabel('Lag Times (ms)')
ylabel('Cross-correlation')
axis([-5,5,-.4,1.1])
hold on
plot(1000*lags,real(R(:,2)))
plot(1000*lags,real(R(:,3)))
hold off
```



Plot Cross-Correlation of Several Signals

Load a musical sound signal with a sample rate is 8192 hertz. Then, duplicate the signal two times and introduce time delays between the two signals and the reference signal. Estimate the time delays and plot the cross-correlation function between all pairs of signals.

```
load handel;
dt = 1/Fs;
refsig = y;
```

Create three delayed versions of the signal.

```
delay1 = -5.7*dt;
delay2 = 10.2*dt;
sig1 = delayseq(refsig,delay1,Fs);
sig2 = delayseq(refsig,delay2,Fs);
```

Cross-correlate all signals with the other signal.

```
[tau_est,R,lags] = gccphat([refsig,sig1,sig2],Fs);
```

Show the time delays in units of sample interval. The algorithm estimates time delays quantized to the nearest sample interval. Cross-correlation of three signals produce 9 possible time delays, one for each possible signal pair.

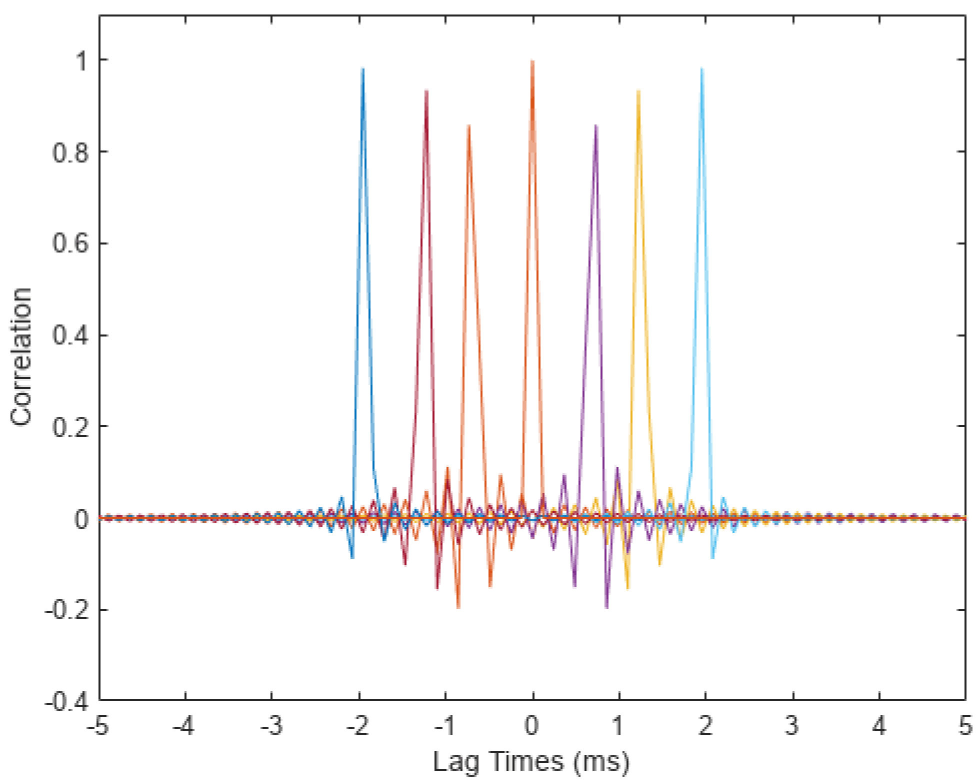
```
disp(tau_est*Fs)
```

```
     0    -6    10     6     0    16   -10   -16     0
```

A signal correlated with itself gives zero lag.

Plot the correlation functions.

```
for n=1:9
    plot(1000*lags,real(R(:,n)))
    if n==1
        hold on
        xlabel('Lag Times (ms)')
        ylabel('Correlation')
        axis([-5,5,-.4,1.1])
    end
end
hold off
```



Input Arguments

sig — Sensor signals

N-by-1 complex-valued column vector | *N*-by-*M* complex-valued matrix

Sensor signals, specified as an *N*-by-1 column vector or an *N*-by-*M* matrix. *N* is the number of time samples and *M* is the number of channels. If `sig` is a matrix, each column is a different channel.

Example: `[0,1,2,3,2,1,0]`

Data Types: `single` | `double`
Complex Number Support: Yes

refsig — Reference sensor signals

N-by-1 complex-valued column vector | *N*-by-*M* complex-valued matrix

Reference signals, specified as an *N*-by-1 complex-valued column vector or an *N*-by-*M* complex-valued matrix. If `refsig` is a column vector, then all channels in `sig` use `refsig` as the reference signal when computing the cross-correlation.

If `refsig` is a matrix, then the size of `refsig` must match the size of `sig`. The `gccphat` function computes the cross-correlation between corresponding channels in `sig` and `refsig`. The signals can come from different sources.

Example: `[1,2,3,2,1,0,0]`

Data Types: `single` | `double`
Complex Number Support: Yes

fs — Signal sample rate

1 (default) | positive real-valued scalar

Signal sample rate, specified as a positive real-valued scalar. All signals should have the same sample rate. Sample rate units are in hertz.

Example: 8000

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

tau — Time delay

1-by-*K* real-valued row vector

Time delay, returned as a 1-by-*K* real-valued row vector. The value of *K* depends upon the input argument syntax.

- When a reference signal, `refsig`, is used, the value of *K* equals the column dimension of `sig`, *M*. Each entry in `tau` specifies the estimated delay for the corresponding signal pairs in `sig` and `refsig`.
- When no reference signal is used, the value of *K* equals the square of the column dimension of `sig`, M^2. Each entry in `tau` specifies the estimated delay for the corresponding signal pairs in `sig`.

Units are seconds.

R — Cross-correlation between signals
(2*N*-1)-by-*K* complex-valued matrix

Cross-correlation between signals at different sensors, returned as a (2*N*-1)-by-*K* complex-valued matrix.

- When a reference signal, `refsig`, is used, the value of *K* equals the column dimension of `sig`, *M*. Each column is the cross-correlation between the corresponding signal pairs in `sig` and `refsig`.
- When no reference signal is used, the value of *K* equals the square of the column dimension of `sig`, M^2. Each column is the cross-correlation between the corresponding signal pairs in `sig`.

`lag` — Cross-correlation lag times
(2*N*-1) real-valued column vector

Correlation lag times, returned as a *(2N-1)* real-valued column vector. Each row of `lag` contains the lag time for the corresponding row of `R`. Lag values are constrained to be multiples of the sampling interval. Lag units are in seconds.

More About

Generalized Cross-Correlation

You can use generalized cross-correlation to estimate the time difference of arrival of a signal at two different sensors.

A model of a signal emitted by a source and received at two sensors is given by:

$$r_1(t) = s(t) + n_1(t)$$
$$r_2(t) = s(t - D) + n_2(t)$$

where *D* is the time difference of arrival (*TDOA*), or time lag, of the signal at one sensor with respect to the arrival time at a second sensor. You can estimate the time delay by finding the time lag that maximizes the cross-correlation between the two signals.

From the TDOA, you can estimate the broadside arrival angle of the plane wave with respect to the line connecting the two sensors. For two sensors separated by distance *L*, the broadside arrival angle, “Broadside Angles”, is related to the time lag by

$$\sin\beta = \frac{c\tau}{L}$$

where *c* is the propagation speed in the medium.

A common method of estimating time delay is to compute the cross-correlation between signals received at two sensors. To identify the time delay, locate the peak in the cross-correlation. When the signal-to-noise ratio (SNR) is large, the correlation peak, τ, corresponds to the actual time delay *D*.

$$R(\tau) = E\{r_1(t)r_2(t+\tau)\}$$
$$\widehat{D} = \underset{\tau}{\operatorname{argmax}} R(\tau)$$

When the correlation function is more sharply peaked, performance improves. You can sharpen a cross correlation peak using a weighting function that whitens the input signals. This technique is called generalized cross-correlation (GCC). One particular weighting function normalizes the signal spectral density by the spectrum magnitude, leading to the generalized cross-correlation phase transform method (*GCC-PHAT*).

$$S(f) = \int_{\infty}^{\infty} R(\tau) e^{-i2\pi f\tau} d\tau$$

$$\tilde{R}(\tau) = \int_{\infty}^{\infty} \frac{S(f)}{|S(f)|} e^{+i2\pi f\tau} df$$

$$\tilde{D} = \underset{\tau}{\operatorname{argmax}}\, \tilde{R}(\tau)$$

If you use just one sensor pair, you can only estimate the broadside angle of arrival. However, if you use multiple pairs of non-collinear sensors, for example, in a URA, you can estimate the arrival azimuth and elevation angles of the plane wave using least-square estimation. For *N* sensors, you can write the delay time τ_{kj} of a signal arriving at the k^{th} sensor with respect to the j^{th} sensor by

$$c\tau_{kj} = -\left(\vec{x}_k - \vec{x}_j\right) \cdot \vec{u}$$

$$\vec{u} = \cos\alpha\sin\theta\,\hat{i} + \sin\alpha\sin\theta\,\hat{j} + \cos\theta\,\hat{k}$$

where *u* is the unit propagation vector of the plane wave. The angles α and θ are the azimuth and elevation angles of the propagation vector. All angles and vectors are defined with respect to the local axes. You can solve the first equation using least-squares to yield the three components of the unit propagation vector. Then, you can solve the second equation for the azimuth and elevation angles.

Version History

Introduced in R2015b

References

[1] Knapp, C. H. and G.C. Carter, "The Generalized Correlation Method for Estimation of Time Delay." *IEEE Transactions on Acoustics, Speech and Signal Processing.* Vol. ASSP-24, No. 4, Aug 1976.

[2] G. C. Carter, "Coherence and Time Delay Estimation." *Proceedings of the IEEE.* Vol. 75, No. 2, Feb 1987.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phased.GCCEstimator`

global2localcoord

Convert global to local coordinates

Syntax

```
lclCoord = global2localcoord(gCoord)
lclCoord = global2localcoord(gCoord,option)
lclCoord = global2localcoord( ___ ,localOrigin)
lclCoord = global2localcoord( ___ ,localAxes)
```

Description

`lclCoord = global2localcoord(gCoord)` converts the global rectangular coordinates `gCoord` to the local rectangular coordinates `lclCoord`. In this syntax, the global coordinate origin is located at (0, 0, 0) and the coordinate axes are the unit vectors in the *x*, *y*, and *z* directions.

`lclCoord = global2localcoord(gCoord,option)` converts global coordinates to local coordinates using the coordinate transformation type `option`.

`lclCoord = global2localcoord( ___ ,localOrigin)` specifies the origin of the local coordinate system `localOrigin`. Use this syntax with any of the input arguments in previous syntaxes.

`lclCoord = global2localcoord( ___ ,localAxes)` specifies the axes of the local coordinate system `localAxes`. Use this syntax with any of the input arguments in previous syntaxes.

Examples

Convert Global Coordinates to Local Coordinates

Convert the global rectangular coordinates, (0, 1, 0), to local rectangular coordinates. The local coordinate origin is (1, 1, 1).

```
lclCoord = global2localcoord([0;1;0],"rr",[1;1;1])
```

```
lclCoord = 3×1

    -1
     0
    -1
```

Convert global spherical coordinates to local rectangular coordinates.

```
lclCoord = global2localcoord([45;45;50],"sr",[50;50;50])
```

```
lclCoord = 3×1

  -25.0000
  -25.0000
```

```
  -14.6447
```

Convert Two Vectors Between Local and Global Coordinates

Convert two vectors from global to local coordinates using the `global2localcoord` function. Then convert them back from local to global coordinates using the `local2globalcoord` function.

Start with two vectors in global coordinates, (0, 1, 0) and (1, 1, 1). The local coordinate origins are (1, 5, 2) and (-4, 5, 7), respectively.

```
gCoord = [0 1;1 1;0 1]
```

```
gCoord = 3×2

     0     1
     1     1
     0     1
```

```
lclOrig = [1 -4;5 5;2 7];
```

Construct two rotation matrices using the rotation functions.

```
lclAxes(:,:,1) = rotz(45)*roty(-15);
lclAxes(:,:,2) = roty(45)*rotx(35);
```

Convert the vectors from global coordinates to local coordinates.

```
lclCoord = global2localcoord(gCoord,"rr",lclOrig,lclAxes)
```

```
lclCoord = 3×2

   -3.9327    7.7782
   -2.1213   -3.6822
   -1.0168    1.7151
```

Convert the vectors from local coordinates back to global coordinates.

```
gCoord1 = local2globalcoord(lclCoord,"rr",lclOrig,lclAxes)
```

```
gCoord1 = 3×2

   -0.0000    1.0000
    1.0000    1.0000
         0    1.0000
```

Input Arguments

gCoord — Global coordinates in rectangular or spherical coordinate form
3-by-*N* matrix

Global coordinates in rectangular or spherical coordinate form, specified as a 3-by-*N* matrix. Each column represents one set of global coordinates.

If the coordinates are in rectangular form, each column contains the (*x*,*y*,*z*) components. Units are in meters.

If the coordinates are in spherical form, each column contains (*az*,*el*,*r*) components. *az* is the azimuth angle on page 2-186 in degrees, *el* is the elevation angle on page 2-186 in degrees, and *r* is the radius in meters.

The origin of the global coordinate system is assumed to be (0, 0, 0). The global system axes are the standard unit basis vectors in three-dimensional space, (1, 0, 0), (0, 1, 0), and (0, 0, 1).

Data Types: `double`

`option` — Type of coordinate transformation
`"rr"` (default) | string scalar | character vector

Type of coordinate transformation, specified as a string scalar or character vector. Specify one of the following values.

Value	Transformation
`"rr"` or `'rr'`	Global rectangular to local rectangular
`"rs"` or `'rs'`	Global rectangular to local spherical
`"sr"` or `'sr'`	Global spherical to local rectangular
`"ss"` or `'ss'`	Global spherical to local spherical

Data Types: `string` | `char`

`localOrigin` — Origin of local coordinate system
`[0;0;0]` (default) | 3-by-*N* matrix

Origin of the local coordinate system, specified as a 3-by-*N* matrix containing the rectangular coordinates of the local coordinate system origin with respect to the global coordinate system. *N* must match the number of columns of `gCoord`. Each column represents a separate origin. Alternatively, you can specify `localOrigin` as a 3-by-1 vector. If you do so, `localOrigin` expands to a 3-by-*N* matrix with identical columns.

Data Types: `double`

`localAxes` — Axes of local coordinate system
`[1 0 0;0 1 0;0 0 1]` (default) | 3-by-3-by-*N* array

Axes of the local coordinate system, specified as a 3-by-3-by-*N* array. Each page contains a 3-by-3 matrix representing axes for a different local coordinate system. The columns of the 3-by-3 matrices specify the local *x*, *y*, and *z* axes in rectangular form with respect to the global coordinate system. Alternatively, you can specify `localAxes` as a single 3-by-3 matrix. If you do so, `localAxes` expands to a 3-by-3-by-*N* array with identical 3-by-3 matrices. The default value is the identity matrix.

Data Types: `double`

Output Arguments

`lclCoord` — Local coordinates in rectangular or spherical coordinate form
3-by-*N* matrix

Local coordinates in rectangular or spherical coordinate form, returned as a 3-by-*N* matrix. The dimensions of `lclCoord` match the dimensions of `gCoord`.

Data Types: `double`

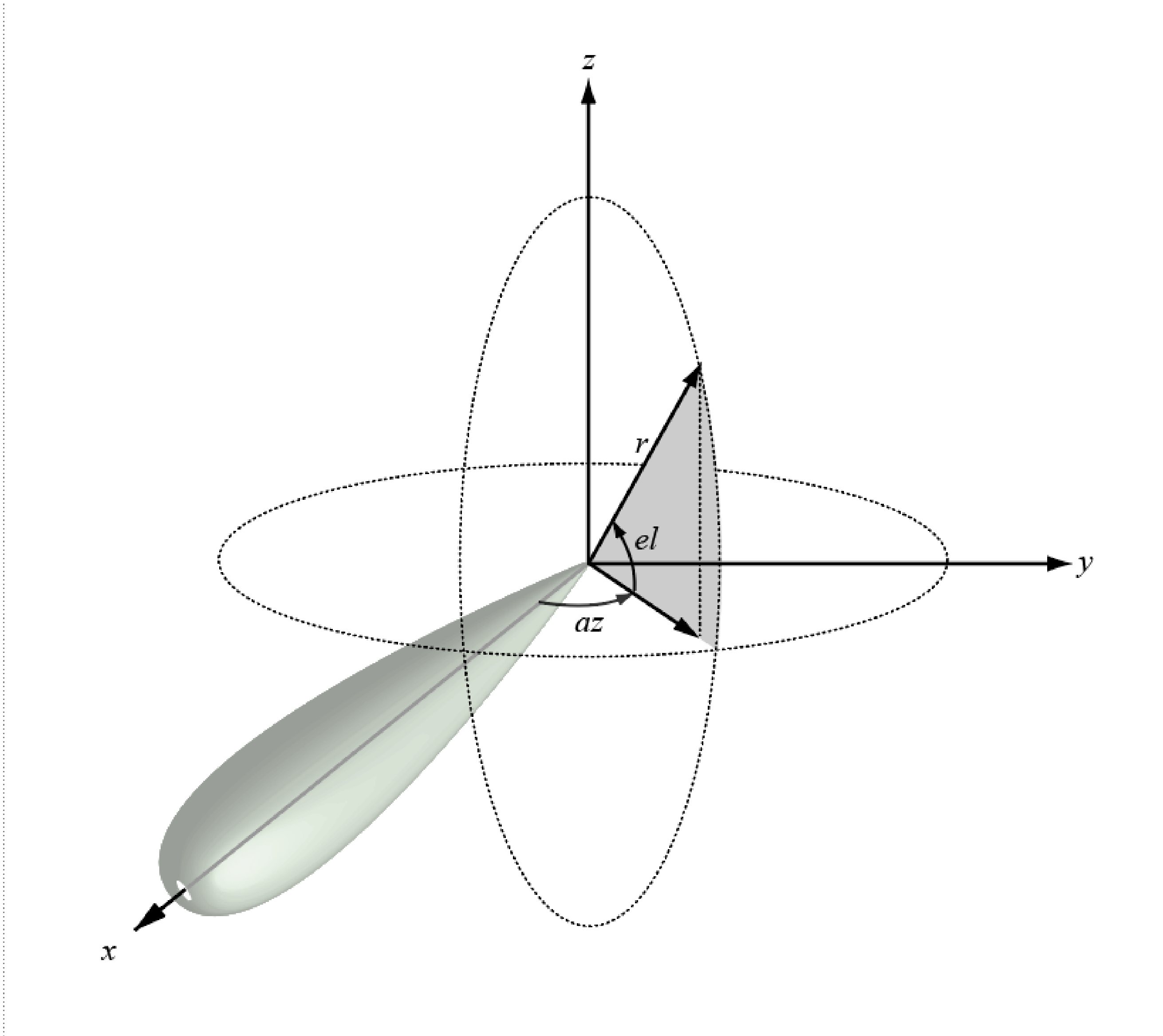
More About

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy*-plane. The angle is positive from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive toward the positive *z*-axis from the *xy*-plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

This figure illustrates the azimuth and elevation angles of a direction vector.



Version History

Introduced in R2011a

References

[1] Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice in C*, 2nd Ed. Reading, MA: Addison-Wesley, 1995.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`local2globalcoord` | `uv2azel` | `phitheta2azel` | `azel2uv` | `azel2phitheta` | `rangeangle`

Topics
"Global and Local Coordinate Systems"

iscoprime

Check coprime relation

Syntax

```
iscp = iscoprime(x)
[iscp,ispcp,pidx,pgcd] = iscoprime(x)
```

Description

`iscp = iscoprime(x)` returns `true` if all elements of `x` are coprime and `false` if two or more elements of `x` have a greatest common divisor (`gcd`) greater than 1.

`[iscp,ispcp,pidx,pgcd] = iscoprime(x)` checks if pairs of elements of `x` have a greatest common divisor greater than 1. This syntax also returns the indices of all pairs of elements of `x` and the greatest common divisor of each pair.

Examples

Coprime Array Elements

Create an array `x` whose elements are 9 = 3 × 3, 15 = 3 × 5, and 25 = 5 × 5. Verify that all elements of `x` are coprime.

```
x = [9 15 25];
```

```
iscp = iscoprime(x)
```

```
iscp = logical
   1
```

Verify that at least one pair of elements of `x` has a greatest common divisor greater than 1. Output the pairs and their greatest common divisors.

```
[~,ispcp,pidx,pgcd] = iscoprime(x)
```

```
ispcp = logical
   0

pidx = 2×3

     1     1     2
     2     3     3

pgcd = 1×3

     3     1     5
```

Input Arguments

x — Input array
row vector of positive integers

Input array, specified as a row vector of positive integers.

Example: `[21 36 49]`

Data Types: `single` | `double`

Output Arguments

`iscp` — True if all elements are coprime
logical scalar

True if all elements are coprime, returned as a logical scalar.

`ispcp` — True if elements are pairwise coprime
logical scalar

True if all elements are pairwise coprime, returned as a logical scalar. `ispcp` is `true` if `x` has no two elements whose greatest common divisor is greater than 1. `ispcp` is `false` if any two elements of `x` have as greatest common divisor a number greater than 1.

`pidx` — Array pair indices
two-row matrix

Array pair indices, returned as a two-row matrix. `pidx` has $\binom{n}{2} = \frac{1}{2}n(n-1)$ columns. Each column of `pidx` specifies the indices of a pair of elements in `x`.

`pgcd` — Pair greatest common divisors
row vector

Pair greatest common divisors, returned as a row vector with a number of elements equal to the number of columns of `pidx`. Each element of `pgcd` is the greatest common divisor of the two elements of `x` identified by the indices in the corresponding column of `pidx`.

Version History

Introduced in R2021a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`coincidence` | `crt`

lcmvweights

Narrowband linearly constrained minimum variance (LCMV) beamformer weights

Syntax

```
wt = lcmvweights(constr,resp,cov)
```

Description

`wt = lcmvweights(constr,resp,cov)` returns narrowband linearly-constrained minimum variance (LCMV) beamformer weights, `wt`, for a phased array. When applied to the elements of the array, these weights steer the response of the array toward a specific arrival direction or set of directions. LCMV beamforming requires that the beamformer response to signals from a direction of interest are passed with specified gain and phase delay. However, power from interfering signals and noise from all other directions is minimized. Additional constraints may be imposed to specifically nullify output power coming from known directions. The constraints are contained in the matrix, `constr`. Each column of `constr` represents a separate constraint vector. The desired response to each constraint is contained in the response vector, `resp`. The argument `cov` is the sensor spatial covariance matrix. All elements in the sensor array are assumed to be isotropic.

Examples

LCMV Beamformer with Nulls at -40 and 20 Degrees

Construct a 10-element half-wavelength-spaced line array. Then, compute the LCMV weights for a desired arrival direction of 0 degrees azimuth. Impose three direction constraints: a null at -40 degrees, a unit desired response in the arrival direction 0 degrees, and another null at 20 degrees. The sensor spatial covariance matrix includes two signals arriving from -60 and 60 degrees and -10 dB isotropic white noise.

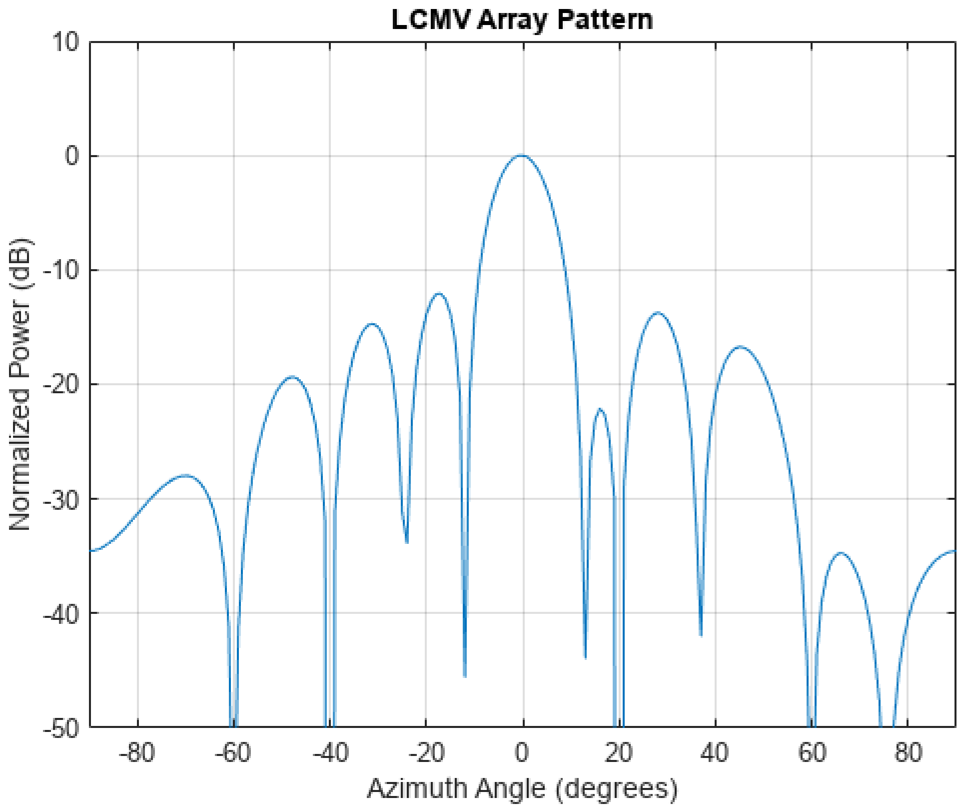
```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
sv = steervec(elementPos,[-40 0 20]);
resp = [0 1 0]';
Sn  = sensorcov(elementPos,[-60 60],db2pow(-10));
```

Compute the beamformer weights.

```
w = lcmvweights(sv,resp,Sn);
```

Plot the array pattern for the computed weights.

```
vv = steervec(elementPos,[-90:90]);
plot([-90:90],mag2db(abs(w'*vv)))
grid on
axis([-90,90,-50,10]);
xlabel('Azimuth Angle (degrees)');
ylabel('Normalized Power (dB)');
title('LCMV Array Pattern');
```



The above figure shows that maximum gain is attained at 0 degrees as expected. In addition, the constraints impose nulls at -40 and 20 degrees and these can be seen in the plot. The nulls at -60 and 60 degrees arise from the fundamental property of the LCMV beamformer of suppressing the power contained in the two plane waves that contributed to the sensor spatial covariance matrix.

Input Arguments

`constr` — Constraint matrix

N-by-*K* complex-valued matrix

Constraint matrix specified as a complex-valued, *N*-by-*K*, complex-valued matrix. In this matrix *N* represents the number of elements in the sensor array while *K* represents the number of constraints. Each column of the matrix specifies a constraint on the beamformer weights. The number of *K* must be less than or equal to *N*.

Example: [0, 0, 0; .1, .2, .3; 0,0,0]

Data Types: `double`
Complex Number Support: Yes

`resp` — Desired response

K-by-1 complex-valued column vector.

Desired response specified as complex-valued, *K*-by-1 column vector where *K* is the number of constraints. The value of each element in the vector is the desired response to the constraint specified in the corresponding column of `constr`.

Example: [45;0]

Data Types: `double`
Complex Number Support: Yes

`cov` — Sensor spatial covariance matrix
N-by-*N* complex-valued matrix

Sensor spatial covariance matrix specified as a complex-valued, *N*-by-*N* matrix. In this matrix, *N* represents the number of sensor elements. The covariance matrix consists of the variances of the element data and the covariance between sensor elements. It contains contributions from all incoming signals and noise.

Example: [45;0]

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`wt` — Beamformer weights
N-by-1 complex-valued vector

Beamformer weights returned as an *N*-by-1, complex-valued vector. In this vector, *N* represents the number of elements in the array.

More About

Linear-Constrained Minimum Variance Beamformers

The LCMV beamformer computes weights that minimize the total output power of an array but that are subject to some constraints (see Van Trees [1], p. 527). In order to steer the response of the array to a particular arrival direction, weights are chosen to produce unit gain when applied to the steering vector for that direction. This requirement can be thought of as a constraint on the weights. Additional constraints may be applied to nullify the array response to signals from other arrival directions such as those containing noise sources. Let $(az_1,el_1),(az_2,el_2),\ldots,(az_K,el_K)$ be the set of directions for which a constraint is to be imposed. Each direction has a corresponding steering vector, $\mathbf{c}_k$, and the response of the array to that steering vector is given by $\mathbf{c}_k^H\mathbf{w}$. The transpose conjugate of a vector is denoted by the superscript symbol H. A constraint is imposed when a desired response is required when the beamformer weights act on a steering vector, $\mathbf{c}_k$,

$$\mathbf{c}_k^H\mathbf{w} = r_k$$

This response could be specified as unity to allow the array to pass through the signal from a certain direction. It could be zero to nullify the response from that direction. All the constraints can be collected into a single matrix, C, and all the response into a single column vector, $\mathbf{R}$. This allows the constraints to be represented together in matrix form

$$C^H\mathbf{w} = \mathbf{R}$$

The LCMV beamformer chooses weights to minimize the total output power

$$P = \mathbf{w}^H S\mathbf{w}$$

subject to the above constraints. S denotes the sensor spatial correlation matrix. The solution to the power minimization is

$$\mathbf{w} = S^{-1}C\left(C^H S^{-1} C\right)^{-1}\mathbf{R}$$

and its derivation can be found in [2].

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York, NY: Wiley-Interscience, 2002.

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`cbfweights` | `mvdrweights` | `sensorcov` | `steervec` | `phased.LCMVBeamformer`

local2globalcoord

Convert local to global coordinates

Syntax

```
gCoord = local2globalcoord(lclCoord)
gCoord = local2globalcoord(lclCoord,option)
gCoord = local2globalcoord( ___ ,localOrigin)
gCoord = local2globalcoord( ___ ,localAxes)
```

Description

`gCoord = local2globalcoord(lclCoord)` converts the local rectangular coordinates `lclCoord` to the global rectangular coordinates `gCoord`.

`gCoord = local2globalcoord(lclCoord,option)` converts local coordinates to global coordinates using the coordinate transformation type `option`.

`gCoord = local2globalcoord( ___ ,localOrigin)` specifies the origin of the local coordinate system `localOrigin`. Use this syntax with any of the input arguments in previous syntaxes.

`gCoord = local2globalcoord( ___ ,localAxes)` specifies the axes of the local coordinate system `localAxes`. Use this syntax with any of the input arguments in previous syntaxes.

Examples

Convert Local Rectangular Coordinates to Global Rectangular Coordinates

Convert local rectangular coordinates to global rectangular coordinates. The local coordinate origin is (1, 1, 1).

```
globalcoord = local2globalcoord([0;1;0],"rr",[1;1;1])
```

```
globalcoord = 3×1

     1
     2
     1
```

Convert Local Spherical Coordinates to Global Rectangular Coordinates

Convert local spherical coordinates to global rectangular coordinates.

```
globalcoord = local2globalcoord([30;45;4],"sr")
```

```
globalcoord = 3×1
```

```
    2.4495
    1.4142
    2.8284
```

Convert Two Vectors Between Local and Global Coordinates

Convert two vectors from global to local coordinates using the `global2localcoord` function. Then convert them back from local to global coordinates using the `local2globalcoord` function.

Start with two vectors in global coordinates, (0, 1, 0) and (1, 1, 1). The local coordinate origins are (1, 5, 2) and (-4, 5, 7), respectively.

```
gCoord = [0 1;1 1;0 1]
```

```
gCoord = 3×2

     0     1
     1     1
     0     1
```

```
lclOrig = [1 -4;5 5;2 7];
```

Construct two rotation matrices using the rotation functions.

```
lclAxes(:,:,1) = rotz(45)*roty(-15);
lclAxes(:,:,2) = roty(45)*rotx(35);
```

Convert the vectors from global coordinates to local coordinates.

```
lclCoord = global2localcoord(gCoord,"rr",lclOrig,lclAxes)
```

```
lclCoord = 3×2

   -3.9327    7.7782
   -2.1213   -3.6822
   -1.0168    1.7151
```

Convert the vectors from local coordinates back to global coordinates.

```
gCoord1 = local2globalcoord(lclCoord,"rr",lclOrig,lclAxes)
```

```
gCoord1 = 3×2

   -0.0000    1.0000
    1.0000    1.0000
         0    1.0000
```

Input Arguments

`lclCoord` — Local coordinates in rectangular or spherical coordinate form
3-by-*N* matrix

Local coordinates in rectangular or spherical coordinate form, specified as a 3-by-*N* matrix. Each column represents one set of local coordinates.

If the coordinates are in rectangular form, each column contains the (*x*,*y*,*z*) components. Units are in meters.

If the coordinates are in spherical form, each column contains (*az*,*el*,*r*) components. *az* is the azimuth angle on page 2-186 in degrees, *el* is the elevation angle on page 2-186 in degrees, and *r* is the radius in meters.

Data Types: `double`

`option` — Type of coordinate transformation
`"rr"` (default) | string scalar | character vector

Type of coordinate transformation, specified as a string scalar or character vector. Specify one of the following values.

Value	Transformation
`"rr"` or `'rr'`	Local rectangular to global rectangular
`"rs"` or `'rs'`	Local rectangular to global spherical
`"sr"` or `'sr'`	Local spherical to global rectangular
`"ss"` or `'ss'`	Local spherical to global spherical

Data Types: `string` | `char`

`localOrigin` — Origin of local coordinate system
`[0;0;0]` (default) | 3-by-*N* matrix

Origin of the local coordinate system, specified as a 3-by-*N* matrix containing the rectangular coordinates of the local coordinate system origin with respect to the global coordinate system. *N* must match the number of columns of `gCoord`. Each column represents a separate origin. Alternatively, you can specify `localOrigin` as a 3-by-1 vector. If you do so, `localOrigin` expands to a 3-by-*N* matrix with identical columns.

Data Types: `double`

`localAxes` — Axes of local coordinate system
`[1 0 0;0 1 0;0 0 1]` (default) | 3-by-3-by-*N* array

Axes of the local coordinate system, specified as a 3-by-3-by-*N* array. Each page contains a 3-by-3 matrix representing axes for a different local coordinate system. The columns of the 3-by-3 matrices specify the local *x*, *y*, and *z* axes in rectangular form with respect to the global coordinate system. Alternatively, you can specify `localAxes` as a single 3-by-3 matrix. If you do so, `localAxes` expands to a 3-by-3-by-*N* array with identical 3-by-3 matrices. The default value is the identity matrix.

Data Types: `double`

Output Arguments

`gCoord` — Global coordinates in rectangular or spherical coordinate form
3-by-*N* matrix

Global coordinates in rectangular or spherical coordinate form, returned as a 3-by-*N* matrix. The dimensions of `gCoord` match the dimensions of `lclCoord`. The origin of the global coordinate system is assumed to be (0, 0, 0). The global system axes are the standard unit basis vectors in three-dimensional space, (1, 0, 0), (0, 1, 0), and (0, 0, 1).

Data Types: `double`

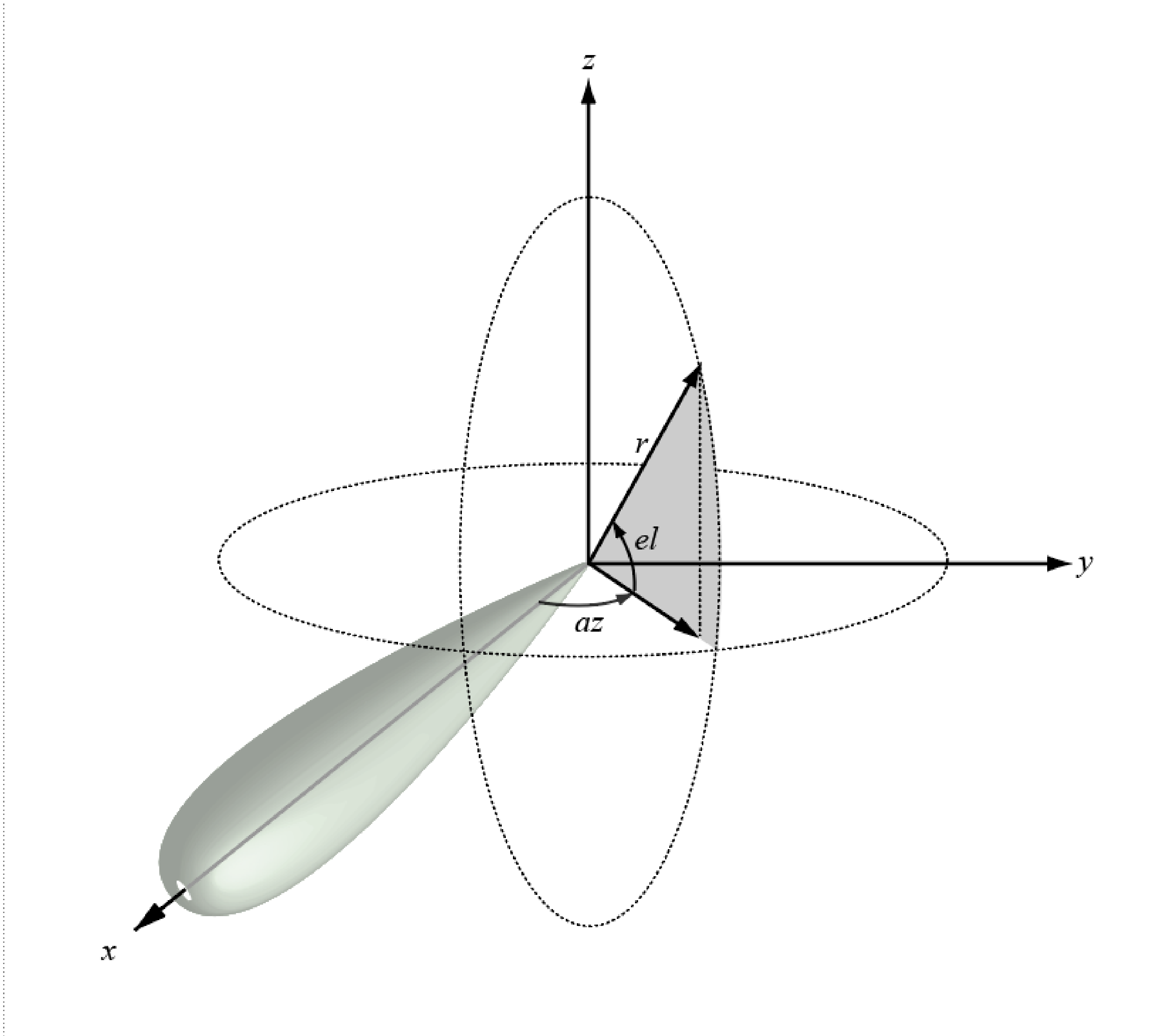
More About

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy*-plane. The angle is positive from the *x*-axis toward the *y*-axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive toward the positive *z*-axis from the *xy*-plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

This figure illustrates the azimuth and elevation angles of a direction vector.



Version History

Introduced in R2011a

References

[1] Foley, J. D., A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice in C*, 2nd Ed. Reading, MA: Addison-Wesley, 1995.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`global2localcoord` | `uv2azel` | `phitheta2azel` | `azel2uv` | `azel2phitheta` | `rangeangle`

Topics

“Global and Local Coordinate Systems”

mdltest

Dimension of signal subspace

Syntax

```
nsig = mdltest(X)
nsig = mdltest(X,'fb')
```

Description

`nsig = mdltest(X)` estimates the number of signals, `nsig`, present in a snapshot of data, `X`, that impinges upon the sensors in an array. The estimator uses the Minimum Description Length (MDL) test. The input argument, `X`, is a complex-valued matrix containing a time sequence of data samples for each sensor. Each row corresponds to a single time sample for all sensors.

`nsig = mdltest(X,'fb')` estimates the number of signals. Before estimating, it performs forward-backward averaging on the sample covariance matrix constructed from the data snapshot, `X`. This syntax can use any of the input arguments in the previous syntax.

Examples

Estimate the Signal Subspace Dimensions for Two Arriving Signals

Construct a data snapshot of two plane waves arriving at a half-wavelength-spaced uniform line array having 10 elements. The plane waves arrive from 0° and –25° azimuth, both with elevation angles of 0°. Assume the signals arrive in the presence of additive noise that is both temporally and spatially Gaussian white. For each signal, the SNR is 5 dB. Take 300 samples to build a 300-by-10 data snapshot. Then, solve for the number of signals using `mdltest`.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25];
x = sensorsig(elementPos,300,angles,db2pow(-5));
Nsig = mdltest(x)
```

```
Nsig = 2
```

The result shows that the number of signals is two, as expected.

Estimate the Signal Subspace Dimensions Using Forward-Backward Averaging

Construct a data snapshot for two plane waves arriving at a half-wavelength-spaced uniform line array with 10 elements. Correlated plane waves arrive from 0° and 10° azimuth, both with elevation angles of 0°. Assume the signals arrive in the presence of additive noise that is both temporally and spatially Gaussian white noise. For each signal, the SNR is 10 dB. Take 300 samples to build a 300-by-10 data snapshot. Then, solve for the number of signals using `mdltest`.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 10];
ncov = db2pow(-10);
scov = [1 .5]'*[1 .5];
x = sensorsig(elementPos,300,angles,ncov,scov);
Nsig = mdltest(x)
```

```
Nsig = 1
```

This result shows that `mdltest` cannot determine the number of signals correctly when the signals are correlated.

Now, try the forward-backward smoothing option.

```
Nsig = mdltest(x,'fb')
```

```
Nsig = 2
```

The addition of forward-backward smoothing yields the correct number of signals.

Input Arguments

X — Data snapshot

complex-valued K-by-N matrix

Data snapshot, specified as a complex-valued, K-by-N matrix. A snapshot is a sequence of time-samples taken simultaneous at each sensor. In this matrix, K represents the number of time samples of the data, while N represents the number of sensor elements.

Example: [–0.1211 + 1.2549i, 0.1415 + 1.6114i, 0.8932 + 0.9765i;]

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`nsig` — Dimension of signal subspace

non-negative integer

Dimension of signal subspace, returned as a non-negative integer. The dimension of the signal subspace is the number of signals in the data.

More About

Estimating the Number of Sources

AIC and MDL tests

Direction finding algorithms such as MUSIC and ESPRIT require knowledge of the number of sources of signals impinging on the array or equivalently, the dimension, d, of the signal subspace. The Akaike Information Criterion (AIC) and the Minimum Description Length (MDL) formulas are two frequently-used estimators for obtaining that dimension. Both estimators assume that, besides the signals, the data contains spatially and temporally white Gaussian random noise. Finding the number of sources is equivalent to finding the multiplicity of the smallest eigenvalues of the sampled spatial covariance matrix. The sample spatial covariance matrix constructed from a data snapshot is used in place of the actual covariance matrix.

A requirement for both estimators is that the dimension of the signal subspace be less than the number of sensors, N, and that the number of time samples in the snapshot, K, be much greater than N.

A variant of each estimator exists when forward-backward averaging is employed to construct the spatial covariance matrix. Forward-backward averaging is useful for the case when some of the sources are highly correlated with each other. In that case, the spatial covariance matrix may be ill conditioned. Forward-backward averaging can only be used for certain types of symmetric arrays, called centro-symmetric arrays. Then the forward-backward covariance matrix can be constructed from the sample spatial covariance matrix, S, using $S_{FB} = S + JS^*J$ where J is the exchange matrix. The exchange matrix maps array elements into their symmetric counterparts. For a line array, it would be the identity matrix flipped from left to right.

All the estimators are based on a cost function

$$L_d(d) = K(N-d)\ln\left\{\frac{\frac{1}{N-d}\sum_{i=d+1}^{N}\hat{\lambda}_i}{\left\{\prod_{i=d+1}^{N}\hat{\lambda}_i\right\}^{\frac{1}{N-d}}}\right\}$$

plus an added penalty term. The value λ_i represent the smallest $(N–d)$ eigenvalues of the spatial covariance matrix. For each specific estimator, the solution for d is given by

- AIC

$$\hat{d}_{AIC} = \underset{d}{\operatorname{argmin}}\,\{L_d(d) + d(2N-d)\}$$

- AIC for forward-backward averaged covariance matrices

$$\hat{d}_{AIC:FB} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + \frac{1}{2}d(2N-d+1)\right\}$$

- MDL

$$\hat{d}_{MDL} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + \frac{1}{2}(d(2N-d)+1)\ln K\right\}$$

- MDL for forward-backward averaged covariance matrices

$$\hat{d}_{MDL\,FB} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + \frac{1}{4}d(2N-d+1)\ln K\right\}$$

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`aictest` | `espritdoa` | `rootmusicdoa` | `spsmooth`

mvdrweights

Minimum variance distortionless response (MVDR) beamformer weights

Syntax

```
wt = mvdrweights(pos,ang,cov)
wt = mvdrweights(pos,ang,nqbits)
```

Description

`wt = mvdrweights(pos,ang,cov)` returns narrowband minimum variance distortionless response (MVDR) beamformer weights for a phased array. When applied to the elements of an array, the weights steer the response of a sensor array in a specific arrival direction or set of directions. The `pos` argument specifies the sensor positions of the array. The `ang` argument specifies the azimuth and elevation angles of the desired response directions. `cov` is the sensor spatial covariance matrix between sensor elements. The output argument, `wt`, is a matrix contains the beamformer weights for each sensor and each direction. Each column of `wt` contains the weights for the corresponding direction specified in `ang`. All elements in the sensor array are assumed to be isotropic.

`wt = mvdrweights(pos,ang,nqbits)` returns quantized narrowband MVDR beamformer weights when the number of phase shifter bits is set to `nqbits`.

Examples

MVDR Beamformer with Arrival Directions of 30 and 45 Degrees

Construct a 10-element, half-wavelength-spaced line array. Choose two arrival directions of interest - one at 30° azimuth and the other at 45° azimuth. Assume both directions are at 0° elevation. Compute the MVDR beamformer weights for each direction. Specify a sensor spatial covariance matrix that contains signals arriving from -60° and 60° and noise at -10 dB.

Set up the array and sensor spatial covariance matrix.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
Sn  = sensorcov(elementPos,[-60 60],db2pow(-10));
```

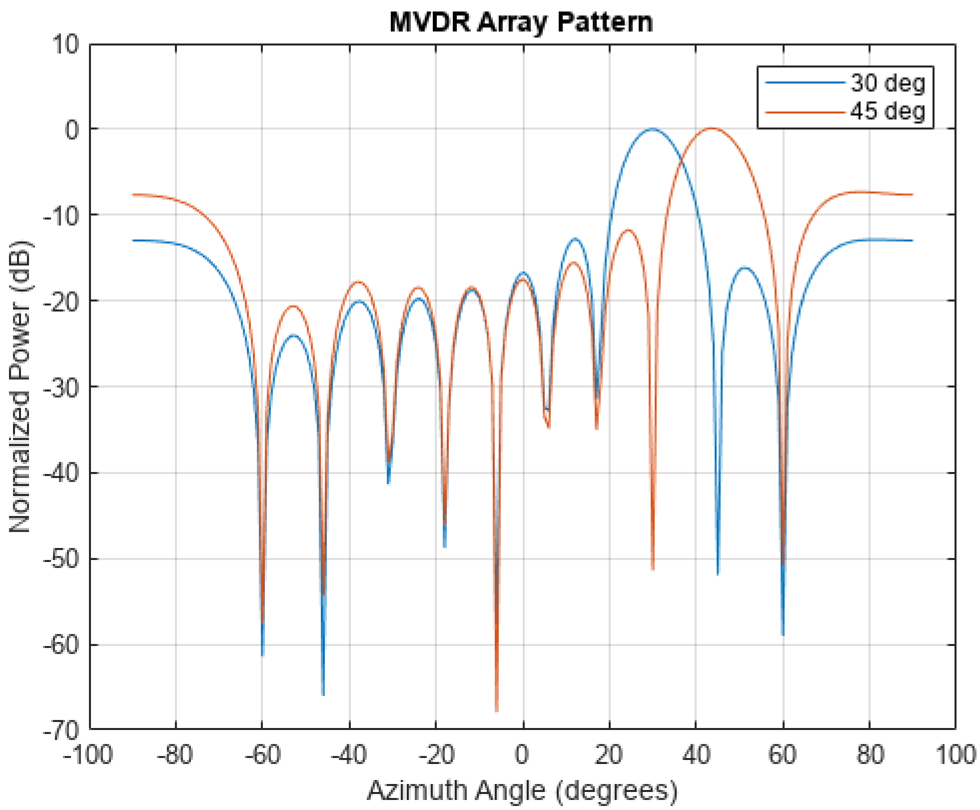
Solve for the MVDR beamformer weights.

```
w = mvdrweights(elementPos,[30 45],Sn);
```

Plot the two MVDR array patterns.

```
plotangl = -90:90;
vv = steervec(elementPos,plotangl);
plot(plotangl,mag2db(abs(w'*vv)))
grid on
xlabel('Azimuth Angle (degrees)');
ylabel('Normalized Power (dB)');
```

```
legend('30 deg','45 deg');
title('MVDR Array Pattern')
```



The figure shows plots for each beamformer direction. One plot has the expected maximum gain at 30 degrees and the other at 45 degrees. The nulls at -60 and 60 degrees arise from the fundamental property of the MVDR beamformer of suppressing power in all directions except for the arrival direction.

Quantized Weights in MVDR Beamformer

Construct a 10-element, half-wavelength-spaced line array. Choose the arrival direction of interest to be 18.5° azimuth and 10° elevation. Compute the MVDR beamformer weights and then compute the weights for 3-bit quantization. Specify a sensor spatial covariance matrix that contains signals arriving from -60° and 60° and noise at -10 dB.

Set up the array and the sensor spatial covariance matrix.

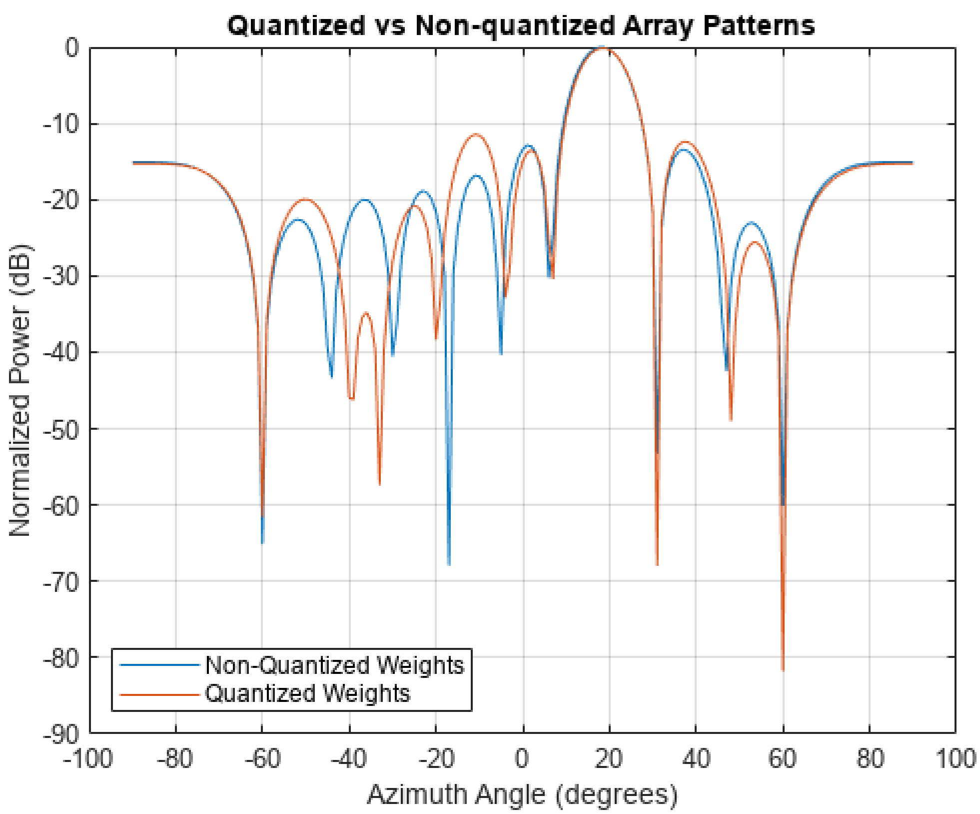
```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
SN  = sensorcov(elementPos,[-60 60],db2pow(-10));
```

Solve for the MVDR beamformer weights with and without quantization.

```
w = mvdrweights(elementPos,[18.5;10],SN);
wq = mvdrweights(elementPos,[18.5;10],SN,3);
```

Plot both MVDR array patterns.

```
plotangl = -90:90;
vv = steervec(elementPos,plotangl);
plot(plotangl,mag2db(abs(w'*vv)))
hold on
plot(plotangl,mag2db(abs(wq'*vv)))
grid on
xlabel('Azimuth Angle (degrees)')
ylabel('Normalized Power (dB)')
legend('Non-Quantized Weights','Quantized Weights','Location','SouthWest');
title('Quantized vs Non-quantized Array Patterns')
hold off
```



Input Arguments

pos — Positions of array sensor elements

1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array, specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero.

When `pos` is a 2-by-*N* matrix, it represents the *(y,z)*-coordinates of the sensor elements of a planar array. This array is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. When `pos` is a 3-by-*N* matrix, then the array can have an arbitrary shape. Sensor positions are in terms of signal wavelength.

Example: `[0,0,0; 0.1,0.4,0.3; 1,1,1]`

Data Types: `double`

`ang` — Beamforming directions
1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Beamforming directions, specified as a 1-by-*M* vector or a 2-by-*M* matrix. In this vector or matrix, *M* represents the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the beamforming direction as `[az;el]`. Angular units are specified in degrees. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the beamforming direction vector onto the *xy* plane. The angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the beamforming direction vector and *xy*-plane. It is positive when measured towards the positive *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero.

Example: `[45;10]`

Data Types: `double`

`cov` — Sensor spatial covariance matrix
N-by-*N* complex-valued matrix

Sensor spatial covariance matrix specified as an *N*-by-*N*, complex-valued matrix. In this matrix, *N* represents the number of sensor elements.

Example: `[5,0.1;0.1,2]`

Data Types: `double`
Complex Number Support: Yes

`nqbits` — Number of phase shifter quantization bits
0 (default) | non-negative integer

Number of bits used to quantize the phase shift in beamformer or steering vector weights, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Example: 5

Output Arguments

`wt` — Beamformer weights
N-by-*M* complex-valued matrix

Beamformer weights returned as a complex-valued, *N*-by-*M* matrix. In this matrix, *N* represents the number of sensor elements of the array while *M* represents the number of beamforming directions. Each column of `wt` corresponds to a beamforming direction specified in `ang`.

More About

Minimum Variance Distortionless Response

MVDR beamformer weights minimize the total array output power while setting the gain in the desired response direction to unity (see Van Trees [1], p. 442). MVDR weights are given by

$$\mathbf{w} = \frac{S^{-1}|\mathbf{v_0}}{\mathbf{v_0}^{H} S^{-1}\mathbf{v_0}}$$

where $\mathbf{v}_0$ is the steering vector corresponding to the desired response direction. S is the spatial covariance matrix. The covariance matrix consists of the variances of the element data and the covariances of the data between the sensor elements. The covariance contains contributions from all incoming signals and noise.

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York, NY: Wiley-Interscience, 2002.

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`steervec` | `cbfweights` | `lcmvweights` | `sensorcov` | `phased.MVDRBeamformer`

minvarweights

Weights for minimum-variance array pattern synthesis

Syntax

```
wts = minvarweights(pos,ang)
wts = minvarweights(pos,ang,cov)
wts = minvarweights( ___ ,MaskAngle=angm)
wts = minvarweights( ___ ,MaskSidelobeLevel=sllm)
wts = minvarweights( ___ ,NullAngle=angn)
```

Description

`wts = minvarweights(pos,ang)` computes the minimum-variance weights `wts` for synthesizing the pattern of a sensor array in the directions specified by `ang`. Array element positions are specified in `pos`. The function optimizes the beamforming weights using a second-order cone programming solver. This function requires Optimization Toolbox™.

`wts = minvarweights(pos,ang,cov)` also specifies the spatial covariance matrix `cov` of the array elements.

`wts = minvarweights( ___ ,MaskAngle=angm)` also specifies angles `angm` at which mask sidelobe levels are defined in the `sllm` argument.

`wts = minvarweights( ___ ,MaskSidelobeLevel=sllm)` also specifies maximum allowable sidelobe levels `sllm` at the angles defined in `angm`.

`wts = minvarweights( ___ ,NullAngle=angn)` also specifies null directions `angn` for the array.

Examples

Compute Optimized ULA Beamforming Weights

Compute optimized beamforming weights of a 31-element half-wavelength spacing ULA in the direction of -30° degree in azimuth. Design the array to keep sidelobe levels less than -23 dB.

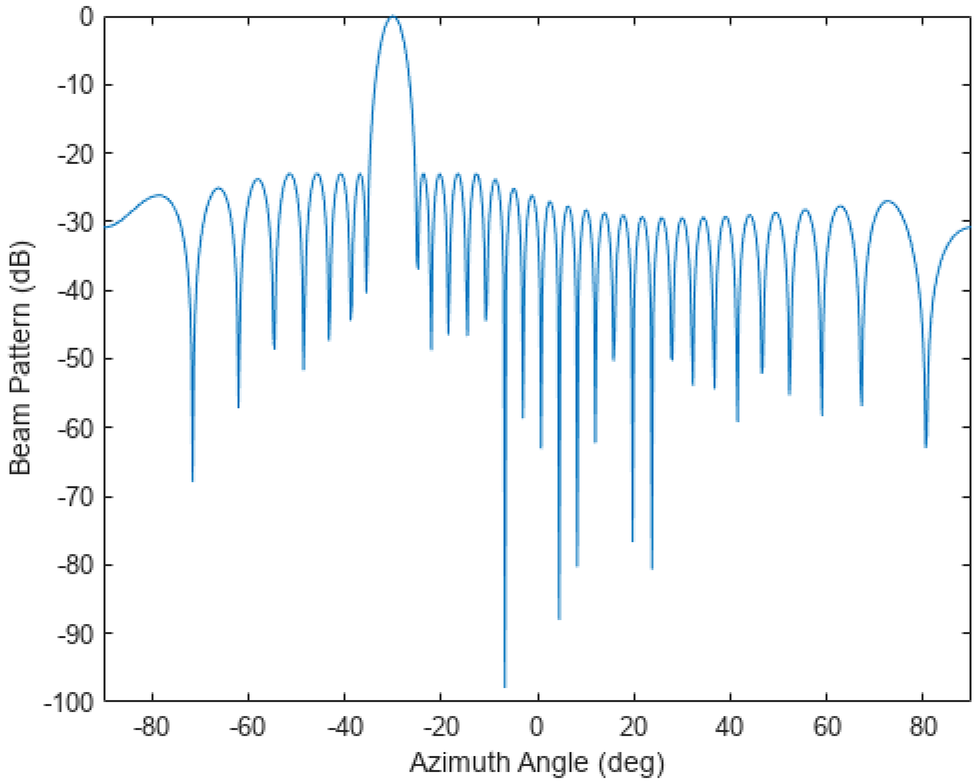
Create the optimized weights.

```
N = 31;
pos = (0:N-1)*0.5;
sll = -23;
wts = minvarweights(pos,-30,MaskSidelobeLevel=sll);
```

Apply the optimized weights and display the array pattern from -90° to $+90^{\circ}$ azimuth.

```
az = -90:.25:90;
pat_opt = arrayfactor(pos,az,wts);
plot(az,mag2db(abs(pat_opt)))
xlabel('Azimuth Angle (deg)')
```

```
ylabel('Beam Pattern (dB)')
xlim([-90,90])
```



Optimized Tapered ULA Weights

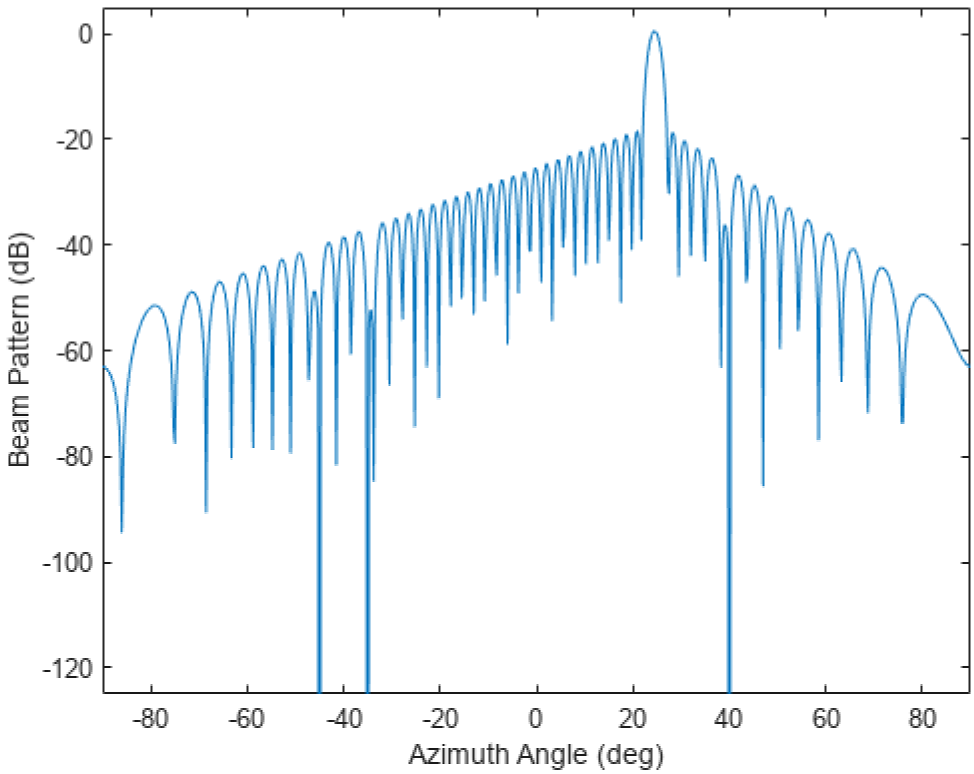
Design an array to have a tapered beampattern, The array is a 51-element half-wavelength spacing ULA steered in the direction of 25° in azimuth. The pattern synthesis goal is to achieve sidelobe levels smaller than a tapered mask decreasing linearly from -18 dB to -55 dB at ±90°. Place nulls at -35°, -45°, and 40° azimuth angle.

```
N = 51;
pos = (0:N-1)*0.5;
ANGmainBeam = 25;
angn = [-35 -45 40];
angm = [-90:.2:22 27:0.2:90];
sllm = [linspace(-55,-18,length(-90:.2:22)) ...
    linspace(-18,-55,length(27:.2:90))];
wts = minvarweights(pos,ANGmainBeam,'MaskAngle',angm, ...
    'MaskSidelobeLevel',sllm,'NullAngle',angn);
```

Apply optimized weights and display the array pattern from -90° to $+90^\circ$ in azimuth.

```
az = -90:.25:90;
pat_opt = arrayfactor(pos,az,wts);
```

```
plot(az,mag2db(abs(pat_opt)))
axis([-90 90 -125 5])
xlabel('Azimuth Angle (deg)')
ylabel('Beam Pattern (dB)')
```



Verify that nulls are placed at -35°, -45°, and 40° azimuth angle.

Input Arguments

pos — Positions of array sensor elements

1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array, specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero. When `pos` is a 2-by-*N* matrix, it represents the *(y,z)*-coordinates of the sensor elements of a planar array. This array is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. When `pos` is a 3-by-*N* matrix, then the array can have an arbitrary shape. Sensor positions are in terms of signal wavelength.

Example: `[0,0,0; 0.1,0.4,0.3; 1,1,1]`

Data Types: `double`

ang — Beamforming directions
1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Beamforming directions, specified as a 1-by-*M* vector or a 2-by-*M* matrix. In this vector or matrix, *M* represents the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the beamforming direction as `[az;el]`. Angular units are specified in degrees. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the beamforming direction vector onto the *xy* plane. The angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the beamforming direction vector and *xy*-plane. It is positive when measured towards the positive *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero.

Example: `[45;10]`

Data Types: `double`

cov — Sensor spatial covariance matrix
`eye(N)` (default) | *N*-by-*N* complex-valued matrix

Sensor spatial covariance matrix, specified as an *N*-by-*N* complex-valued matrix. *N* is the number of array sensor elements.

Example: `[5,0.1;0.1,2]`

Data Types: `double`
Complex Number Support: Yes

angm — Mask angles
`[]` (default) | real-valued 1-by-*K* vector | real-valued 2-by-*K* matrix

Angles at which mask sidelobe levels are defined, specified as a real-valued 1-by-*K* vector or a real-valued 2-by-*K* matrix where *K* is the number of mask sidelobe levels. If `angm` is a 1-by-*K* vector, then it contains the azimuth angles of the mask directions. If `angm` is a 2-by-*K* matrix, each column specifies the direction in the form `[az;el]`. Angle units are in degrees.

Data Types: `double`

`sllm` — Maximum allowable mask sidelobe levels
non-positive scalar (default) | non-positive real-valued 1-by-*K* vector

Maximum allowable mask sidelobe levels, specified as a non-positive scalar or non-positive real-valued 1-by-*K* vector. *K* is the number of mask sidelobe levels. Sidelobe levels are always less then or equal to zero.

- If `sllm` is a scalar, then it contains a uniform mask for all sidelobe levels and `angm` must be empty.
- If `sllm` is a 1-by-*K* vector, then `sllm` and `angm` must have the same number of columns; and `sllm` contains the mask sidelobe levels for corresponding mask angles, `angm`.

An empty `sllm` vector means that there are no constraints on the sidelobe levels. Units are in dB.

Data Types: `double`

angn — Null direction angles
`[]` (default) | real-valued 1-by-*P* vector | real-valued 2-by-*P* matrix

Null direction angles, specified as either a 1-by-*P* vector or a 2-by-*P* matrix where *P* is the number of null directions. If `angn` is a 1-by-*P* vector, then it contains only the azimuth angles of directions. If `angn` is a 2-by-*P* matrix, each column specifies the null direction in the form `[az; el]`. Angle units are in degrees.

Data Types: `double`

Output Arguments

`wts` — Beamformer weights
N-by-*1* complex-valued vector

Beamformer weights, returned as a complex-valued *N*-by-*1* vector. *N* represents the number of sensor elements of the array.

Version History

Introduced in R2022b

References

[1] Lebret, H., and S. Boyd. “Antenna Array Pattern Synthesis via Convex Optimization.” *IEEE Transactions on Signal Processing*, vol. 45, no. 3, Mar. 1997, pp. 526–32. DOI.org (Crossref), https://doi.org/10.1109/78.558465.

[2] Golbon-Haghighi, Mohammad-Hossein, et al. “Design of a Cylindrical Crossed Dipole Phased Array Antenna for Weather Surveillance Radars.” *IEEE Open Journal of Antennas and Propagation*, vol. 2, 2021, pp. 402–11. DOI.org (Crossref), https://doi.org/10.1109/OJAP.2021.3059471.

See Also

`steervec` | `cbfweights` | `lcmvweights` | `sensorcov` | `phased.MVDRBeamformer` | `phased.LCMVBeamformer`

musicdoa

Estimate arrival directions of signals using MUSIC

Syntax

```
doas = musicdoa(covmat,nsig)
[doas,spec,specang] = musicdoa(covmat,nsig)
[ ___ ] = musicdoa(covmat,nsig, ___ ,'ScanAngles',scanangle)
[ ___ ] = musicdoa(covmat,nsig, ___ ,'ElementSpacing',dist)
```

Description

`doas = musicdoa(covmat,nsig)` uses the MUSIC algorithm to estimate the directions of arrival, `doas`, of `nsig` plane waves received on a uniform linear array (ULA). The argument `covmat` is a positive-definite Hermitian matrix representing the sensor covariance matrix. Detected sources appear as peaks in the spatial spectrum. The argument `nsig` is the number of arriving signals. Sensor elements are spaced one-half wavelength apart in units of wavelengths. The function forces exact conjugate symmetry of `covmat` by averaging the covariance matrix with its conjugate transpose.

`[doas,spec,specang] = musicdoa(covmat,nsig)` also returns the spatial spectrum, `spec`, and the `nsig` angles of the spectrum peaks, `specang`.

`[ ___ ] = musicdoa(covmat,nsig, ___ ,'ScanAngles',scanangle)` specifies the grid of broadside angles to search for spectrum peaks.

`[ ___ ] = musicdoa(covmat,nsig, ___ ,'ElementSpacing',dist)` specifies the spacing between array elements.

Examples

Estimate DOA of Multiple Signals Using MUSIC

Calculate the directions of arrival of 3 uncorrelated signals arriving at an 11-element ULA with half-wavelength spacing. Assume the signals are coming from the broadside angles of 0°, –12°, and 85°. The noise at each element is Gaussian white noise and is uncorrelated between elements. The SNR is 5 dB.

Specify the number of ULA elements and the element spacing (in wavelengths).

```
nelem = 11;
d = 0.5;
snr = 5.0;
elementPos = (0:nelem-1)*d;
```

Specify the number of signals and their broadside arrival angles.

```
nsig = 3;
angles = [0.0 -12.0 85.0];
```

Create the sensor covariance matrix.

```
covmat = sensorcov(elementPos,angles,db2pow(-snr));
```

Estimate the broadside arrival angles.

```
doas = musicdoa(covmat,nsig)
```

```
doas = 1×3

    85     0   -12
```

The estimated angles match the specified angles.

Display MUSIC Spectrum of Multiple Signals

Calculate the directions of arrival of 3 uncorrelated signals arriving at an 11-element ULA with half-wavelength spacing. Assume the signals are coming from the broadside angles of 0°, –12°, and 85°. The noise at each element is Gaussian white noise and is uncorrelated between elements. The SNR is 2 dB.

Specify the number of ULA elements and the element spacing (in wavelengths).

```
nelem = 11;
d = 0.5;
snr = 2.0;
elementPos = (0:nelem-1)*d;
```

Specify the number of signals and their broadside arrival angles.

```
nsig = 3;
angles = [0.0 -12.0 85.0];
```

Create the sensor covariance matrix.

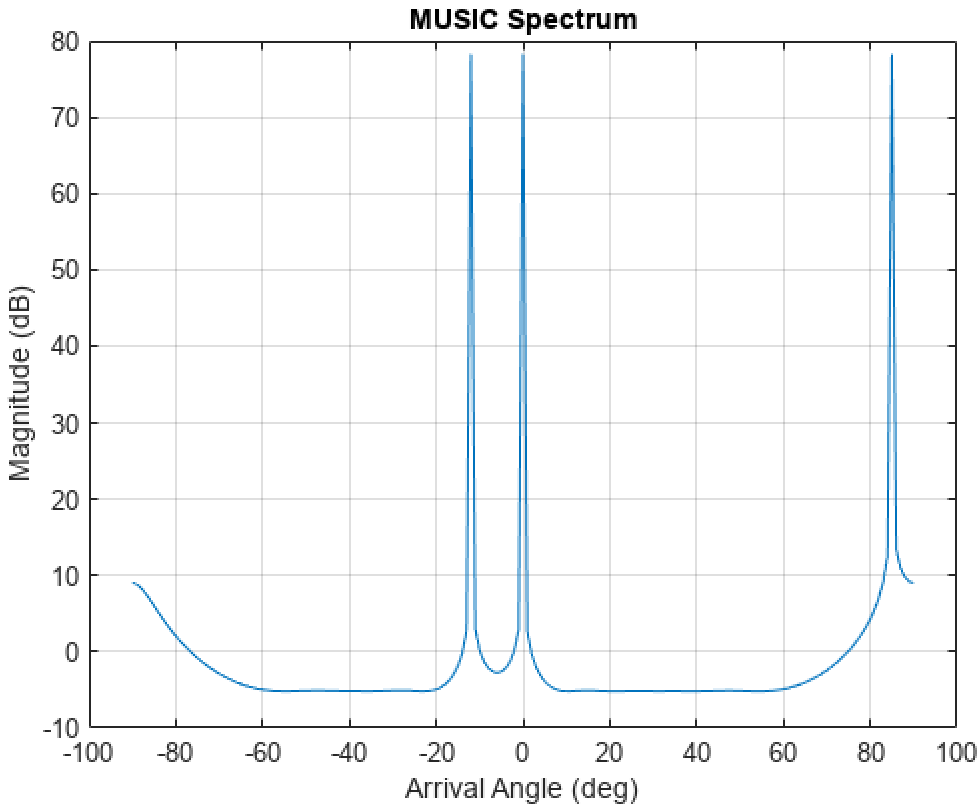
```
covmat = sensorcov(elementPos,angles,db2pow(-snr));
```

Compute the MUSIC spectrum and estimate the broadside arrival angles.

```
[doas,spec,specang] = musicdoa(covmat,nsig);
```

Plot the MUSIC spectrum.

```
plot(specang,10*log10(spec))
xlabel('Arrival Angle (deg)')
ylabel('Magnitude (dB)')
title('MUSIC Spectrum')
grid
```



The estimated angles match the specified angles.

Display MUSIC Spectrum Over Specified Direction Span

Calculate the directions of arrival of 4 uncorrelated signals arriving at an 11-element ULA. The element spacing is 0.5 wavelengths. Assume the signals are coming from the broadside angles of –60.2°, –20.7°, 0.5°, and 84.8°. The noise at each element is Gaussian white noise and is uncorrelated between elements. The SNR is 0 dB.

Specify the number of ULA elements and the element spacing (in wavelengths).

```
nelem = 11;
d = 0.5;
snr = 5.0;
elementPos = (0:nelem-1)*d;
```

Specify the number of signals and their broadside arrival angles.

```
nsig = 4;
angles = [-60.2 -20.7 0.5 84.8];
```

Create the sensor covariance matrix.

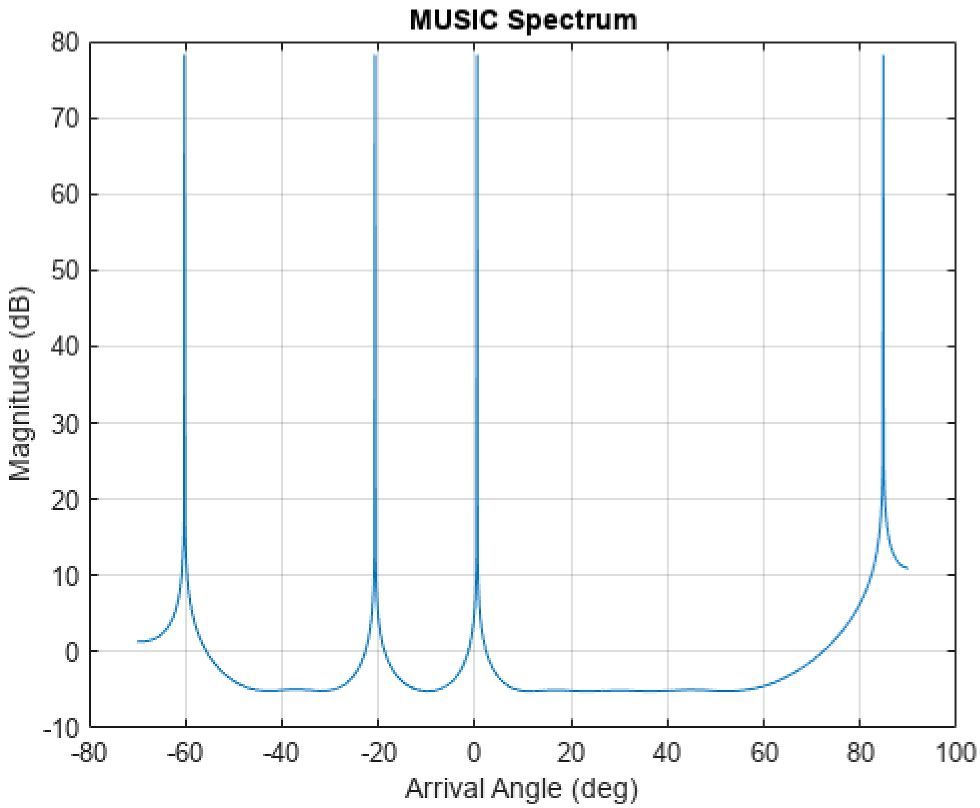
```
covmat = sensorcov(elementPos,angles,db2pow(-snr));
```

Compute the MUSIC spectrum and estimate the broadside arrival angles in the range from -70° to 90° in 0.1° increments.

```
[doas,spec,specang] = musicdoa(covmat,nsig,'ScanAngles',[-70:.1:90]);
```

Plot the MUSIC spectrum.

```
plot(specang,10*log10(spec))
xlabel('Arrival Angle (deg)')
ylabel('Magnitude (dB)')
title('MUSIC Spectrum')
grid
```




```
disp(doas)
```

```
  -60.2000   84.8000    0.5000  -20.7000
```

The estimated angles match the specified angles.

Display MUSIC Spectrum with Specified Element Spacing

Calculate the directions of arrival of 4 uncorrelated signals arriving at an 11-element ULA. The element spacing is 0.4 wavelengths spacing. Assume the signals are coming from the broadside angles of –60°, –20°, 0°, and 85°. The noise at each element is Gaussian white noise and is uncorrelated between elements. The SNR is 0 dB.

Specify the number of ULA elements and the element spacing (in wavelengths).

```
nelem = 11;
d = 0.4;
snr = 0.0;
elementPos = (0:nelem-1)*d;
```

Specify the number of signals and their broadside arrival angles.

```
nsig = 4;
angles = [-60.0 -20.0 0.0 85.0];
```

Create the sensor covariance matrix.

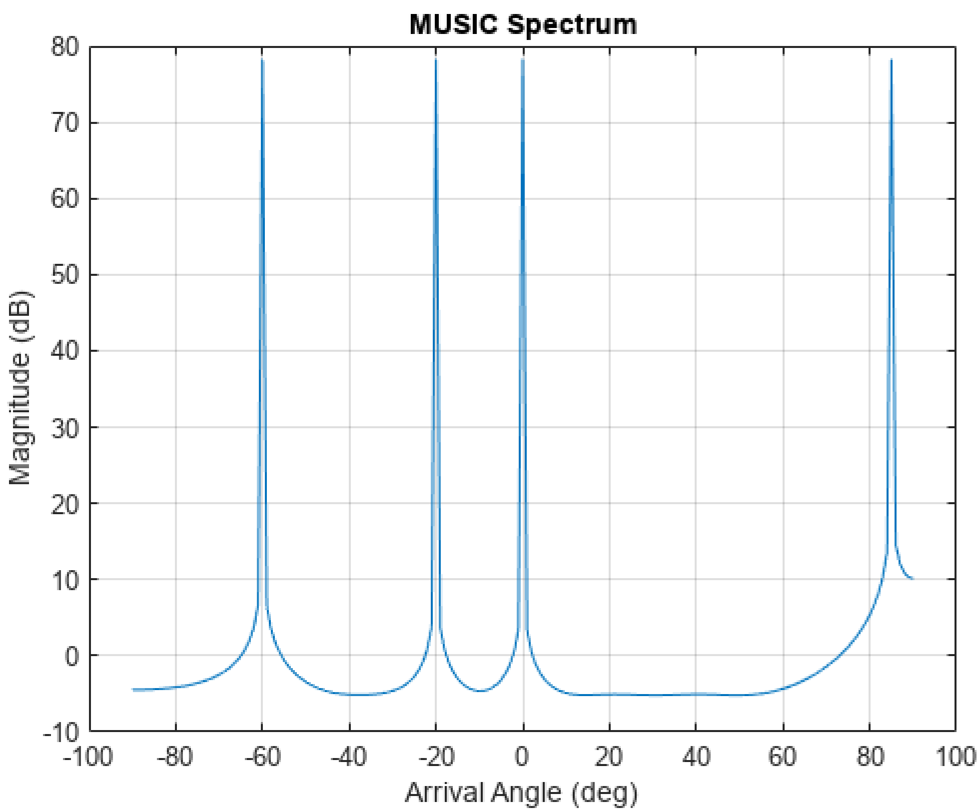
```
covmat = sensorcov(elementPos,angles,db2pow(-snr));
```

Compute the MUSIC spectrum and estimate the broadside arrival angles.

```
[doas,spec,specang] = musicdoa(covmat,nsig,'ElementSpacing',d);
```

Plot the MUSIC spectrum.

```
plot(specang,10*log10(spec))
xlabel('Arrival Angle (deg)')
ylabel('Magnitude (dB)')
title('MUSIC Spectrum')
grid
```



The estimated angles match the specified angles.

Input Arguments

covmat — Sensor covariance matrix
positive-definite complex-valued *M*-by-*M* matrix

Sensor covariance matrix, specified as a complex-valued, positive-definite *M*-by-*M* matrix. The quantity *M* is the number of elements in the ULA array. The function forces Hermiticity property by averaging the matrix and its conjugate transpose.

Data Types: `double`
Complex Number Support: Yes

nsig — Number of arriving signals
positive integer

Number of arriving signals, specified as a positive integer. The number of signals must be smaller than the number of elements in the ULA array.

Example: 2

Data Types: `double`

scanangle — Broadside search angles
`[-90:90]` (default) | real-valued vector

Broadside search angles, specified as a real-valued vector. Angles must lie in the range (–90°,90°) and must be in increasing order.

Example: `[-40:0.5:50]`

Data Types: `double`

dist — Distance between array elements
`0.5` (default) | real-valued positive scalar

Distance between array elements, specified as a real-valued positive scalar.

Example: `0.45`

Data Types: `double`

Output Arguments

doas — Directions of arrival angles
real-valued vector

Directions of arrival angle, returned as a real-valued 1-by-*D* vector, where *D* is the number of arriving signals specified in `nsig`. Angle units are in degrees. Angle values lie in the range specified by `scanangle`.

spec — Spatial spectrum
positive real-valued vector

Spatial spectrum, returned as a positive real-valued vector. The dimension of `spec` equals the dimension of `scanangle`.

specang — Broadside angles of spectrum peaks
real-valued vector

Broadside angle of spectrum, returned as a real-valued vector. The dimension of `specang` equals the dimension of `scanangle`.

Version History

Introduced in R2016b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

Functions

`az2broadside` | `broadside2az` | `db2pow` | `espritdoa` | `rootmusicdoa` | `sensorcov`

Objects

`phased.MUSICEstimator` | `phased.MUSICEstimator2D`

Topics

"Spherical Coordinates"
"MUSIC Super-Resolution DOA Estimation"

nlfmspec2freq

Instantaneous frequency of nonlinear frequency-modulated waveform

Syntax

```
freq = nlfmspec2freq(BW,S)
```

Description

`freq = nlfmspec2freq(BW,S)` returns samples of the instantaneous frequency `freq` for a nonlinear frequency modulated (NLFM) pulse waveform. The waveform sweeps the bandwidth `BW` and has a power spectrum shape `S`. The frequency values in `freq` are found by applying the principle of stationary phase to the power spectrum shape `S`.

Examples

Waveform Derived from Taylor Spectrum Window

Create a nonlinear FM waveform derived from a power spectral density function shaped as a Taylor window with -35 dB sidelobes. The pulse bandwidth is 120 MHz and the pulse duration is10μsec. Generate matched filter coefficients and then apply a matched filter. Plot the resulting matched filter output to display the range sidelobe levels.

```
BW = 120e6;
T = 10e-6;
fs = 10*BW;
```

Generate 200 points of a waveform with instantaneous frequency values defined by a Taylor window. The window has -35 dB sidelobe levels.

```
w = taylorwin(200,4,-35);
freq = nlfmspec2freq(BW,w);
waveform = phased.CustomFMWaveform('SampleRate',fs, ...
    'PulseWidth',T,'FrequencyModulation',freq, ...
    'OutputFormat','Pulses','CoefficientsOutputPort',true);
disp(['Bandwidth = ',num2str(bandwidth(waveform)/1e6),' MHz']);
```

```
Bandwidth = 119.9644 MHz
```

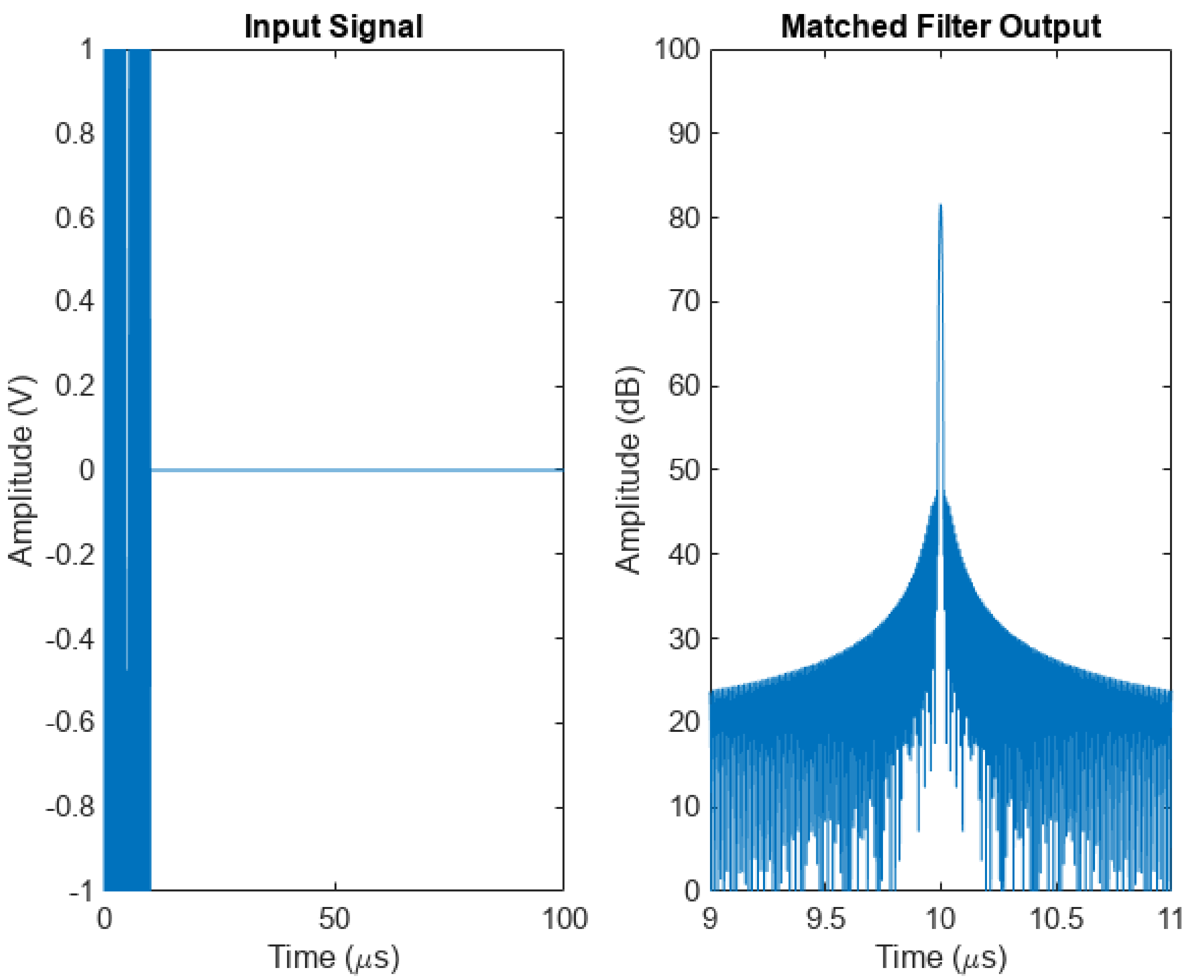
Obtain the matched filter coefficients from the waveform.

```
[wav,coeff] = waveform();
filter = phased.MatchedFilter('CoefficientsSource','Input port');
mfout = filter(wav,coeff);
```

Plot input signal and matched filter output.

```
t = (0:numel(wav)-1)/fs;
figure
subplot(121)
plot(t*1e6,real(wav))
```

```
xlabel('Time (\mus)')
ylabel('Amplitude (V)')
title('Input Signal')

subplot(122)
plot(t*1e6,mag2db(abs(mfout)));
xlabel('Time (\mus)')
ylabel('Amplitude (dB)')
title('Matched Filter Output')
xlim([9 11]);
ylim([0 100]);
```



Input Arguments

BW — Pulse waveform bandwidth
positive scalar

Pulse waveform bandwidth, specified as a positive scalar. Units are in Hz.

Example: `10e3`

Data Types: `double`

S — Power spectrum shape
real-valued vector

Power spectrum shape, specified as a real-valued vector. Units are dimensionless.

Data Types: `double`

Output Arguments

`freq` — Instantaneous waveform frequencies
real-valued row vector

Instantaneous waveform frequencies, returned as a real-valued row vector. The instantaneous frequencies form a single up-sweep. `freq` has the same number of elements as `S`. Units are Hz.

Data Types: `double`

Version History
Introduced in R2023a

References

[1] Collins, T., and P. Atkins. "Nonlinear frequency modulation chirps for active sonar" *IEE Proceedings-Radar, Sonar and Navigation* 146.6 (1999): 312-316.

[2] Levanon, Nadav, and Eli Mozeson. *Radar signals*. John Wiley & Sons, 2004, pp. 92-93.

[3] Doerry, Armin Walter. "Generating nonlinear FM chirp waveforms for radar". No. SAND2006-5856. *Sandia National Laboratories (SNL)*, Albuquerque, NM, and Livermore, CA (United States), 2006.

[4] Cook, C. E. "A class of nonlinear FM pulse compression signals." *Proceedings of the IEEE* 52.11 (1964): 1369-1371.

See Also
`phased.NonlinearFMWaveform` | `phased.CustomFMWaveform`

Topics
"Waveform Analysis Using the Ambiguity Function"

noisefigure

Receiver system noise figure of cascaded stages

Syntax

```
cnf = noisefigure(nf,g)
[cnf,cg] = noisefigure(nf,g)
[cnf,cg,ctemp] = noisefigure(nf,g,reftemp)
```

Description

`cnf = noisefigure(nf,g)` returns the scalar noise figure in decibels of the cascaded stages of a receiver system. The vector `nf` contains the noise figures for the different stages and the vector `g` contains the gains.

`[cnf,cg] = noisefigure(nf,g)` returns the scalar total gain in decibels of the cascaded system.

`[cnf,cg,ctemp] = noisefigure(nf,g,reftemp)` specifies the reference temperature in kelvins and returns the noise temperature of the cascaded receiver stages.

Examples

Cascaded Noise Figure of Receiver System

Compute the cascaded noise figure and total gain of a receiver system. The system has seven stages, with these values:

1. LNA with a noise figure of 1.0 dB and a gain of 15.0 dB
2. RF filter with a noise figure of 0.5 dB and a gain of –0.5 dB
3. Mixer with a noise figure of 5.0 dB and a gain of –7.0 dB
4. IF filter with a noise figure of 1.0 dB and a gain of –1.0 dB
5. IF preamplifier with a noise figure of 0.6 dB and a gain of –15.0 dB
6. IF stages with a noise figure of 1.0 dB and a gain of 20.0 dB
7. Phase detectors with a noise figure of 6.0 dB and a gain of –5.0 dB

```
nf = [1.0 0.5 5.0 1.0 0.6 1.0 6.0];
g = [15.0 -0.5 -7.0 -1.0 15.0 20.0 -5.0];
```

```
[cnf,ng] = noisefigure(nf,g)
```

```
cnf = 1.5252
```

```
ng = 36.5000
```

Compute the noise temperature of the cascaded receiver stages. Specify the reference temperature as 300 K.

```
rtemp = 300;
[~,~,ctemp] = noisefigure(nf,g,rtemp)
ctemp = 426.2281
```

Input Arguments

nf — Stage-by-stage noise figures
vector

Stage-by-stage noise figures in decibels, specified as a vector.

Data Types: double

g — Stage-by-stage gains
vector

Stage-by-stage gains in decibels, specified as a vector.

Data Types: double

reftemp — Reference temperature
290 K (default) | nonnegative scalar

Reference temperature, specified as a nonnegative scalar in kelvins.

Data Types: double

Output Arguments

cnf — Noise figure of cascaded receiver stages
scalar

Noise figure of cascaded receiver stages, returned as a scalar in decibels.

cg — Total gain of cascaded system
scalar

Total gain of cascaded system, returned as a scalar in decibels.

ctemp — Noise temperature of cascaded receiver stages
scalar

Noise temperature of cascaded receiver stages, returned as a scalar in kelvins.

Version History

Introduced in R2021a

References

[1] Barton, David K. *Radar System Analysis and Modeling*. Boston: Artech House, 2005.

See Also

`systemp`

noisepow

Receiver noise power

Syntax

`NPOWER = noisepow(NBW,NF,REFTEMP)`

Description

`NPOWER = noisepow(NBW,NF,REFTEMP)` returns the noise power, `NPOWER`, in watts for a receiver. This receiver has a noise bandwidth `NBW` in hertz, noise figure `NF` in decibels, and reference temperature `REFTEMP` in kelvins.

Examples

Compute Receiver Noise Power with Specified Temperature

Calculate the noise power of a receiver having a noise bandwidth of 10 kHz, a noise figure of 1 dB, and a reference temperature of 300 K.

```
npower = noisepow(10e3,1,300)
```

```
npower = 5.2144e-17
```

Input Arguments

`NBW` — Noise bandwidth of the receiver
positive scalar

Noise bandwidth of the receiver in hertz, specified as a positive scalar. For a superheterodyne receiver, the noise bandwidth is approximately equal to the bandwidth of the intermediate frequency stages [1].

Data Types: `double`

`NF` — Noise figure
nonnegative scalar

Noise figure in decibels, specified as a nonnegative scalar. The noise figure is a dimensionless quantity that indicates how much a receiver deviates from an ideal receiver in terms of internal noise. An ideal receiver only produces the expected thermal noise power for a given noise bandwidth and temperature. A noise figure of 1 (0 dB) indicates that the noise power of a receiver equals the noise power of an ideal receiver. Because an actual receiver cannot exhibit a noise power value less than an ideal receiver, the noise figure is always greater than or equal to 1 (0 dB).

Data Types: `double`

`REFTEMP` — Reference temperature
nonnegative scalar

Reference temperature in kelvins, specified as a nonnegative scalar. This argument specifies the temperature of the receiver. Typical values range from 290 to 300 kelvins.

Data Types: `double`

Output Arguments

`NPOWER` — Noise power
nonnegative scalar

Noise power in watts, returned as a nonnegative scalar. The internal noise power contribution of the receiver to the signal-to-noise ratio.

Version History

Introduced in R2011a

References

[1] Skolnik, M. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phased.ReceiverPreamp`

npwgnthresh

Detection SNR threshold for signal in white Gaussian noise

Syntax

```
snrthresh = npwgnthresh(pfa)
snrthresh = npwgnthresh(pfa,numpulses)
snrthresh = npwgnthresh(pfa,numpulses,dettype)
snrthresh = npwgnthresh(pfa,numpulses,dettype,outscale)
```

Description

`snrthresh = npwgnthresh(pfa)` calculates the "Detection SNR Threshold" on page 2-234 in decibels for detecting a deterministic signal in white Gaussian noise. The detection uses the Neyman-Pearson (NP) decision rule to achieve a specified probability of false alarm, `pfa`. This function uses a square-law detector.

`snrthresh = npwgnthresh(pfa,numpulses)` specifies `numpulses` as the number of pulses used in the pulse integration.

`snrthresh = npwgnthresh(pfa,numpulses,dettype)` specifies `dettype` as the type of detection. A square law detector is used in noncoherent detection.

`snrthresh = npwgnthresh(pfa,numpulses,dettype,outscale)` specifies the output scale.

Examples

Compute Detection Threshold from Pfa

Calculate the detection threshold that achieves a probability of false alarm (pfa) of 0.01. Assume a single pulse with a `real` detection type. Then, verify that this threshold produces a pfa of approximately 0.01. Do this by constructing 10000 real white gaussian noise (wgn) samples and computing the fraction of samples exceeding the threshold.

Compute the threshold from pfa. The detection threshold is expressed as a signal-to-noise ratio in db.

```
pfa = 0.01;
numpulses = 1;
snrthreshold = npwgnthresh(pfa,numpulses,"real")
```

```
snrthreshold = 7.3335
```

Compute fraction of simulated noise samples exceeding the threshold. The noise has unit power with 10000 samples.

```
noisepower = 1;
Ntrial = 10000;
noise = sqrt(noisepower)*randn(1,Ntrial);
```

Express the threshold in amplitude units.

```
threshold = sqrt(noisepower*db2pow(snrthreshold));
calculated_Pfa = sum(noise>threshold)/Ntrial
```

```
calculated_Pfa = 0.0107
```

Detection Threshold Versus Number of Pulses

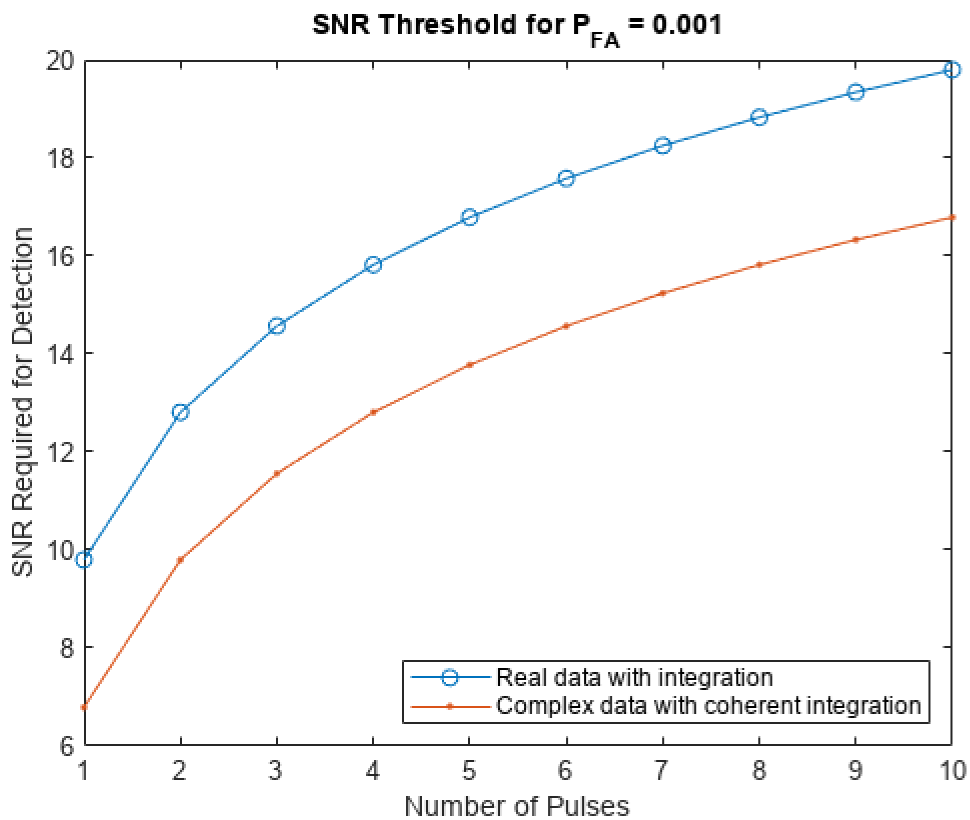
Plot the SNR detection threshold against the number of pulses, for real and complex noise. In each case, the SNR detection threshold is set for a probability of false alarm (pfa) of 0.001.

Compute detection threshold for 1 to 10 pulses of real and complex noise.

```
Npulses = 10;
snrcoh = zeros(1,Npulses);
snrreal = zeros(1,Npulses);
Pfa = 1e-3;
for num = 1:Npulses
    snrreal(num) = npwgnthresh(Pfa,num,"real");
    snrcoh(num)  = npwgnthresh(Pfa,num,"coherent");
end
```

Plot the detection thresholds against number of pulses.

```
plot(snrreal,"o-")
hold on
plot(snrcoh,".-")
hold off

legend("Real data with integration",...
    "Complex data with coherent integration",...
    Location="southeast")
xlabel("Number of Pulses")
ylabel("SNR Required for Detection")
title("SNR Threshold for P_F_A = "+Pfa)
```



Linear Detection Threshold Versus Number of Pulses

Plot the linear detection threshold against the number of pulses, for real and complex data. In each case, the threshold is set for a probability of false alarm of 0.001.

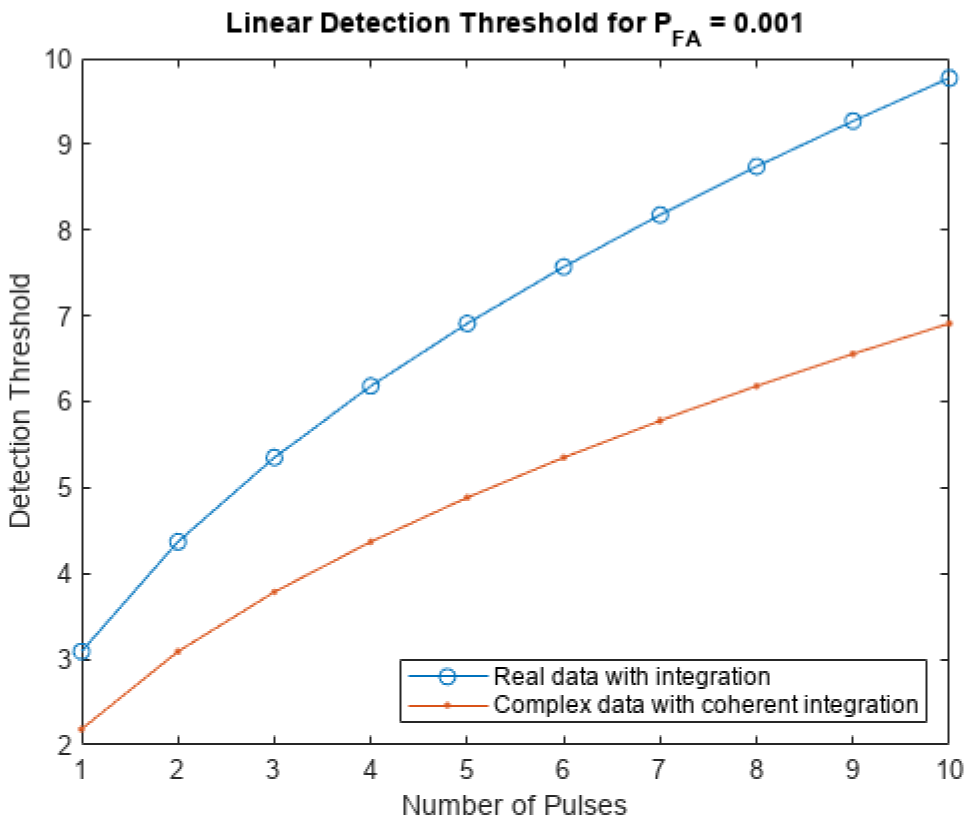
Compute detection threshold for 1 to 10 pulses of real and complex noise.

```
Npulses = 10;
snrcoh = zeros(1,Npulses);
snrreal = zeros(1,Npulses);
Pfa = 1e-3;
for num = 1:Npulses
    snrreal(num) = npwgnthresh(Pfa,num,"real","linear");
    snrcoh(num)  = npwgnthresh(Pfa,num,"coherent","linear");
end
```

Plot the detection thresholds against number of pulses.

```
plot(snrreal,"o-")
hold on
plot(snrcoh,".-")
hold off

legend("Real data with integration",...
    "Complex data with coherent integration",...
```

```
    Location="southeast")
xlabel("Number of Pulses")
ylabel("Detection Threshold")
title("Linear Detection Threshold for P_F_A = "+Pfa)
```



Input Arguments

pfa — Probability of false alarm
scalar in the range `(0, 1)`

Probability of false alarm, specified as a scalar in the range `(0, 1)`.

Data Types: `double`

numpulses — Number of pulses
`1` (default) | positive integer

Number of pulses used in the integration, specified as a positive integer.

Data Types: `double`

dettype — Type of pulse integration
`"noncoherent"` (default) | `"coherent"` | `"real"`

Specifies the type of pulse integration used in the NP decision rule, specified as `"coherent"`, `"noncoherent"`, or `"real"`. `"coherent"` uses magnitude and phase information of complex-valued samples. `"noncoherent"` uses squared magnitudes. `"real"` uses real-valued samples.

Data Types: `char` | `string`

`outscale` — Scale of the output value
`"db"` (default) | `"linear"`

Scale of the output value, specified as `"db"` or `"linear"`. When `outscale` is set to `"linear"`, the returned threshold represents amplitude.

Data Types: `char` | `string`

Output Arguments

`snrthresh` — Detection threshold
scalar

Detection threshold, returned as a scalar. The detection threshold is expressed in terms of the signal-to-noise ratio in decibels or in linear units if `outscale` is set to `"linear"`. The relationship between the linear threshold and the threshold in dB is

$$T_{dB} = 20\log_{10}T_{lin}$$

More About

Detection SNR Threshold

The output of `npwgnthresh` determines the detection threshold required to achieve a particular Pfa.

The threshold increases when pulse integration is used in the receiver. This threshold is not the single sample SNR that is used as an input to `rocsnr` or as the output of `rocpfa`, `albersheim`, and `shnidman`. For any fixed Pfa, you can decrease the single sample SNR required to achieve a particular Pd when pulse integration is used in the receiver. See “Signal Detection in White Gaussian Noise” and “Source Localization Using Generalized Cross Correlation” for examples of how to use `npwgnthresh` in a detection system.

Detection in Real-Valued White Gaussian Noise

This function is designed for the detection of a nonzero mean in a sequence of Gaussian random variables. The function assumes that the random variables are independent and identically distributed, with zero mean. The linear detection threshold λ for an NP detector is

$$\frac{\lambda}{\sigma} = \sqrt{2N}\,\mathrm{erfc}^{-1}(2P_{fa})$$

This threshold can also be expressed as a signal-to-noise ratio in decibels

$$10\log_{10}\left(\frac{\lambda^2}{\sigma^2}\right) = 10\log_{10}\left(2N\left(\mathrm{erfc}^{-1}(2P_{fa})\right)^2\right)$$

In these equations

- σ^2 is the variance of the white Gaussian noise sequence
- N is the number of samples
- $erfc^{-1}$ is the inverse of the complementary error function
- P_{fa} is the probability of false alarm

Note For probabilities of false alarm greater than or equal to 1/2, the formula for detection threshold as SNR is invalid because erfc^{-1} is less than or equal to zero for values of its argument greater than or equal to one. In that case, use the linear output of the function invoked by setting `outscale` to`'linear'`.

Detection in Complex-Valued White Gaussian Noise (Coherent Samples)

The NP detector for complex-valued signals is similar to that discussed in “Source Localization Using Generalized Cross Correlation”. In addition, the function makes these assumptions:

- The variance of the complex-valued Gaussian random variable is divided equally among the real and imaginary parts.
- The real and imaginary parts are uncorrelated.

Under these assumptions, the linear detection threshold for an NP detector is

$$\frac{\lambda}{\sigma} = \sqrt{N}\,\mathrm{erfc}^{-1}(2P_{fa})$$

and expressed as a signal-to-noise ratio in decibels is:

$$10\log_{10}\left(\frac{\lambda^2}{\sigma^2}\right) = 10\log_{10}\left(N\left(\mathrm{erfc}^{-1}(2P_{fa})\right)^2\right)$$

Note For probabilities of false alarm greater than or equal to 1/2, the formula for detection threshold as SNR is invalid because erfc^{-1} is less than or equal to zero for when its argument is greater than or equal to one. In that case, select linear output for the function by setting `outscale` to `'linear'`.

Detection of Noncoherent Samples in White Gaussian Noise

For noncoherent samples in white Gaussian noise, detection of a nonzero mean leads to a square-law detector. For a detailed derivation, see [2], pp. 324–329.

The linear detection threshold for the noncoherent NP detector is:

$$\frac{\lambda}{\sigma} = \sqrt{P^{-1}(N, 1 - P_{fa})}$$

The threshold expressed as a signal-to-noise ratio in decibels is:

$$10\log_{10}\left(\frac{\lambda^2}{\sigma^2}\right) = 10\log_{10}P^{-1}(N, 1 - P_{fa})$$

where $P^{-1}(x, y)$is the inverse of the lower incomplete gamma function, P_{fa} is the probability of false alarm, and N is the number of pulses.

Version History

Introduced in R2011a

References

[1] Kay, S. M. *Fundamentals of Statistical Signal Processing: Detection Theory*. Upper Saddle River, NJ: Prentice Hall, 1998.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`rocpfa` | `rocsnr` | `albersheim` | `shnidman`

Topics

"Signal Detection in White Gaussian Noise"
"Signal Detection Using Multiple Samples"

omphybweights

Compute hybrid beamforming weights using orthogonal matching pursuit

Syntax

```
[wpbb,wprf] = omphybweights(chanmat,ns,ntrf,at)
[wpbb,wprf,wcbb,wcrf] = omphybweights(chanmat,ns,ntrf,at,nrrf,ar)
[ ___ ] = omphybweights(chanmat,ns,ntrf,at,nrrf,ar,npow)
```

Description

`[wpbb,wprf] = omphybweights(chanmat,ns,ntrf,at)` returns the hybrid precoding weights `wpbb` and `wprf` for the channel matrix `chanmat`. The weights are computed using an orthogonal matching pursuit algorithm. `ns` is the number of independent data streams propagated through the channel. `ntrf` specifies the number of RF chains in the transmit array. `at` is a collection of possible analog weights for `wprf`. Together, the precoding weights approximate the optimal full digital precoding weights of `chanmat`.

`[wpbb,wprf,wcbb,wcrf] = omphybweights(chanmat,ns,ntrf,at,nrrf,ar)` also returns the hybrid combining weights `wcbb` and `wcrf`. The input `nrrf` specifies the number of RF chains in the receive array. `ar` is a collection of possible analog weights for `wcrf`.

`[ ___ ] = omphybweights(chanmat,ns,ntrf,at,nrrf,ar,npow)` also specifies the noise power `npow` in each receive antenna element. All subcarriers are assumed to have the same noise power.

Examples

Calculate Effective Channel Matrix

Assume an 8-by-4 MIMO system with four RF chains in a transmit array and two RF chains in a receive array. Show that the hybrid weights can support transmitting two data streams simultaneously.

Specify the positions of the transmitters and receivers in uniform line arrays.

```
txpos = (0:7)*0.5;
rxpos = (0:3)*0.5;
```

Construct the channel matrix.

```
chanmat = scatteringchanmtx(txpos,rxpos,10);
```

Specify the number of transmit and receive RF chains.

```
ntrf = 4;
nrrf = 2;
```

Specify two data streams.

```
ns = 2;
```

Set up the steering vector dictionaries for the transmitting and receiving arrays.

```
txdict = steervec(txpos,-90:90);
rxdict = steervec(rxpos,-90:90);
```

Compute the precoding and combining weights.

```
[Fbb,Frf,Wbb,Wrf] = omphybweights(chanmat,ns,ntrf,txdict,nrrf,rxdict);
```

Calculate the effective channel matrix from the weights. A diagonal effective channel matrix indicates the capability of simultaneous transmission of multiple data streams.

```
chan_eff = Fbb*Frf*chanmat*Wrf*Wbb
```

```
chan_eff = 2×2 complex

   1.0000 - 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   1.0000 + 0.0000i
```

Input Arguments

chanmat — Channel response matrix
complex-valued N_t-by-N_r matrix | complex-valued L-by-N_t-by-N_r array

Channel response matrix, specified as an N_t-by-N_r matrix or a complex-valued L-by-N_t-by-N_r array where

- N_t is the number of elements in the transmitting array.
- N_r is the number of elements in the receiving array.
- L is the number of subcarriers.

Data Types: double
Complex Number Support: Yes

ns — Number of independent data streams
positive integer

Number of independent data streams propagated through the channel, specified as a positive integer.

Data Types: double

ntrf — Number of RF chains in transmit array
positive integer

Number of RF chains in the transmit array, specified as a positive integer.

Data Types: double

at — Collection of possible analog weights
complex-valued N_t-by-P matrix | complex-valued N_t-by-P-by-L array

Collection of possible analog weights for wprf, specified as a complex-valued matrix or array.

- When `chanmat` is an N_t-by-N_r matrix, `at` is a complex-valued N_t-by-P matrix. Each column represents a vector of analog weights.
- When `chanmat` is an L-by-N_t-by-N_r array, `at` is a complex-valued N_t-by-P-by-L array. Each page is an N_t-by-P matrix. Each column represents a vector of analog weights.
- N_t is the number of elements in the transmitting array.
- N_r is the number of elements in the receiving array.
- L is the number of subcarriers.
- P is the number of vectors of analog weights in the collection.

Data Types: `double`
Complex Number Support: Yes

`nrrf` — Number of RF chains in receive array
positive integer

Number of RF chains in the receive array, specified as a positive integer.

Data Types: `double`

`ar` — Collection of possible analog weights
complex-valued N_r-by-Q matrix | complex-valued N_r-by-Q-by-L array

Collection of possible analog weights for `wprf`, specified as a complex-valued matrix or array.

- When `chanmat` is an N_t-by-N_r matrix, `ar` is a complex-valued N_r-by-Q matrix. Each column represents a vector of analog weights.
- When `chanmat` is an L-by-N_t-by-N_r array, `ar` is a complex-valued N_r-by-Q-by-L array. Each page is an N_r-by-Q matrix. Each column represents a vector of analog weights.
- N_t is the number of elements in the transmitting array.
- N_r is the number of elements in the receiving array.
- L is the number of subcarriers.
- Q is the number of vectors of analog weights in the collection.

Data Types: `double`
Complex Number Support: Yes

`npow` — Noise power
0 (default) | nonnegative scalar

Noise power in each receive antenna element, specified as a nonnegative scalar. All subcarriers have the same noise power.

Data Types: `double`

Output Arguments

`wpbb` — Hybrid baseband precoding weights
complex-valued N_s-by-N_{trf} matrix | complex-valued L-by-N_s-by-N_{trf} array

Hybrid baseband precoding weights, returned as a complex-valued matrix or array.

- When `chanmat` is an N_t-by-N_r matrix, `wpbb` is a complex-valued N_s-by-N_{trf} matrix.
- When `chanmat` is an L-by-N_t-by-N_r array, `wpbb` is a complex-valued L-by-N_s-by-N_{trf} array.
- N_s is the number of independent data streams specified by the `ns` argument.
- N_{trf} is the number of RF chains in the transmit array specified by the `ntrf` argument.
- L is the number of subcarriers.

`wprf` — Hybrid RF precoding weights
complex-valued N_{trf}-by-N_t matrix | complex-valued L-by-N_{trf}-by-N_t array

Hybrid RF precoding weights, returned as a complex-valued matrix or array.

- When `chanmat` is an N_t-by-N_r matrix, `wprf` is a complex-valued N_{trf}-by-N_t matrix.
- When `chanmat` is an L-by-N_t-by-N_r array, `wprf` is a complex-valued L-by-N_{trf}-by-N_t array.
- N_t is the number of elements in the transmitting array.
- N_{trf} is the number of RF chains in the transmit array specified by the `ntrf` argument.
- L is the number of subcarriers.

`wcbb` — Hybrid baseband combining weights
complex-valued N_{rrf}-by-N_s matrix | complex-valued L-by-N_{rrf}-by-N_s array

Hybrid baseband combining weights, returned as a complex-valued matrix or array.

- When `chanmat` is an N_t-by-N_r matrix, `wcbb` is a complex-valued N_{rrf}-by-N_s matrix.
- When `chanmat` is an L-by-N_t-by-N_r array, `wcbb` is a complex-valued L-by-N_{rrf}-by-N_s array.
- N_s is the number of independent data streams specified by the `ns` argument.
- N_{rrf} is the number of RF chains in the receive array specified by the `nrrf` argument.
- L is the number of subcarriers.

`wcrf` — Hybrid RF combining weights
complex-valued N_r-by-N_{rrf} | complex-valued L-by-N_r-by-N_{rrf} array

Hybrid RF combining weights, returned as a complex-valued matrix or array.

- When `chanmat` is an N_t-by-N_r matrix, `wcrf` is a complex-valued N_r-by-N_{rrf} matrix.
- When `chanmat` is an L-by-N_t-by-N_r array, `wcrf` is a complex-valued L-by-N_r-by-N_{rrf} array.
- N_t is the number of elements in the transmitting array.
- N_{rrf} is the number of RF chains in the receive array specified by the `nrrf` argument.
- L is the number of subcarriers.

More About

Precoding Weights

The matrix product of the precoding weights `wpbb` x `wprf` approximates the optimal full digital precoding weights of the channel matrix `chanmat`.

Combining Weights

The combining weights `wcbb` and `wcrf`, together with the precoding weights, diagonalize the channel into independent subchannels. The matrix product `wpbb` x `wprf` x `chanmat` x `wcrf`x `wcbb` is approximately diagonal.

Version History

Introduced in R2019b

References

[1] Ayach, Omar El et al. "Spatially Sparse Precoding in Millimeter Wave MIMO Systems" *IEEE Trans on Wireless Communications*. Vol. 13, No. 3, March 2014.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`diagbfweights` | `ompdecomp` | `phased.ScatteringMIMOChannel` | `scatteringchanmtx`

ompdecomp

Decompose signal using orthogonal matching pursuit

Syntax

```
[coeff,dictatom,atomidx,errnorm] = ompdecomp(X,dict)
[coeff,dictatom,atomidx,errnorm] = ompdecomp(X,dict,'MaxSparsity',nm)
[coeff,dictatom,atomidx,errnorm] = ompdecomp(X,dict,'NormWeight',wts)
```

Description

`[coeff,dictatom,atomidx,errnorm] = ompdecomp(X,dict)` computes the decomposition matrices `coeff` and `dictatom` of the signal `X`. The product of the decomposition matrices, `dictatom` x `coeff`, approximates `X`. The atoms in `dictatom` are selected from `dict`. `atomidx` are the indices in `dict` corresponding to `dictatom`. `errnorm` is the decomposition error. The decomposition is based on an *orthogonal matching pursuit* (OMP) algorithm that minimizes the Frobenius norm `||X – dictatom` x `coeff||`.

`[coeff,dictatom,atomidx,errnorm] = ompdecomp(X,dict,'MaxSparsity',nm)` also specifies the maximum sparsity `nm`.

`[coeff,dictatom,atomidx,errnorm] = ompdecomp(X,dict,'NormWeight',wts)` minimizes the weighted Frobenius norm $||wts^{1/2}(X - dictatom \times coeff)||$ using the weights `wts`.

Examples

Decompose ULA Beamsteering Weights

Given a set of optimal, full-digital, beamforming weights for an 8-element uniform linear array, decompose the weights into a product of analog and digital beamforming weights. Assume two RF chains. Show that the combined weights achieve a performance similar to the optimal weights.

Specify the optimal, full-digital, beamforming weights.

```
wopt = steervec((0:7)*0.5,[20 -40]);
```

Create a dictionary of steering vectors.

```
stvdict = steervec((0:7)*0.5,-90:90);
```

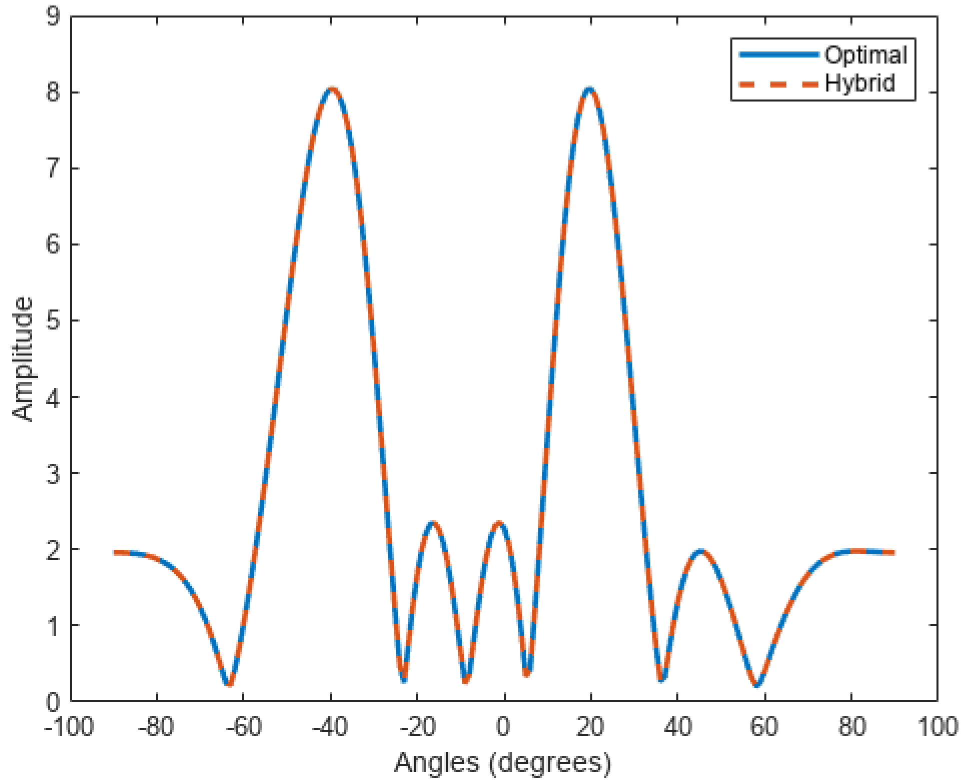
Perform the decomposition using OMP. Set the maximum sparsity to two.

```
[wbb,wrf,wdictidx,normerr] = ompdecomp(wopt,stvdict,'MaxSparsity',2);
```

Compare the beam patterns derived from the optimal weights and the hybrid weights. The plot shows that the decomposition of `wopt` into `wrf` and `wbb` is almost exact.

```
plot(-90:90,abs(sum(wopt'*stvdict)),'-', ...
    -90:90,abs(sum((wrf*wbb)'*stvdict)),'--','LineWidth',2)
xlabel('Angles (degrees)')
```

```
ylabel('Amplitude')
legend('Optimal','Hybrid')
```



Input Arguments

X — Input data
complex-valued N-by-N_c matrix

Input data to be decomposed, specified as a complex-valued N-by-N_c matrix.

Data Types: `double`
Complex Number Support: Yes

dict — Dictionary of atoms
complex-valued matrix

Dictionary of atoms, specified as a complex-valued matrix. The function uses a subset of atoms from the dictionary to construct the data.

Data Types: `double`
Complex Number Support: Yes

nm — Maximum sparsity
`1` (default) | positive integer

Maximum sparsity of the decomposition, specified as a positive integer. The decomposition stops when the sparsity of `nm` is achieved.

Example: 5

Dependencies

Use this argument with the syntax specifying `'MaxSparsity'`.

Data Types: `double`

`wts` — Norm weights

N-by-*N* identity matrix (default) | complex-valued *N*-by-*N* matrix

Norm weights used by OMP to minimize the weighted Frobenius norm of `||wts`$^{1/2}$ x (`X` – `dictatom` x `coeff`)`||`, specified as a complex-valued *N*-by-*N* matrix.

Example: 5

Dependencies

Use this argument with the syntax specifying `'NormWeight'`.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`coeff` — Coefficients of basis atoms

N_s-by-N_c matrix

Coefficients of basis atoms, returned as an N_s-by-N_c matrix. The rows represent the coefficients for the corresponding atoms in `dictatom`. N_s represents the number of atoms selected from the dictionary and is a measure of signal sparsity.

Data Types: `double`
Complex Number Support: Yes

`dictatom` — Signal basis atoms

N-by-N_s matrix

Signal basis atoms, returned as an N-by-N_s matrix. The columns are the atoms forming the basis of the signal. These atoms are a subset of the dictionary specified in `dict`. N_s represents the number of selected atoms and is a measure of signal sparsity.

Data Types: `double`
Complex Number Support: Yes

`atomidx` — Indices of selected atoms

integer-valued length-N_s row vector

Indices of the atoms selected from the dictionary `dict`, returned as a length-N_s row vector where `dict(:,atomidx) = dictatom`.

Data Types: `double`

`errnorm` — Norm of decomposition error

0 (default) | nonnegative scalar

Norm of the decomposition error, returned as a nonnegative scalar.

Data Types: `double`

More About

Hybrid Beamforming Weights

In the context of hybrid beamforming, the `coeff` argument represents digital weights. `dictatom` represents analog weights and `dict` is a collection of steering vectors that can be used as analog weights.

Version History

Introduced in R2019b

References

[1] Ayach, Omar El et al. "Spatially Sparse Precoding in Millimeter Wave MIMO Systems" *IEEE Trans on Wireless Communications*. Vol. 13, No. 3, March 2014.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`omphybweights` | `diagbfweights` | `phased.ScatteringMIMOChannel` | `scatteringchanmtx`

pambgfun

Periodic ambiguity function

Syntax

```
pafmag = pambgfun(X,fs)
[pafmag,delay,doppler] = pambgfun(X,fs)
[pafmag,delay,doppler] = pambgfun(X,fs,P)

[pafmag,delay] = pambgfun( ___ ,'Cut','Doppler')
[pafmag,delay] = pambgfun( ___ ,'Cut','Doppler','CutValue',V)
[pafmag,doppler] = pambgfun( ___ ,'Cut','Delay')
[pafmag,doppler] = pambgfun( ___ ,'Cut','Delay','CutValue',V)
[pafmag,delay,doppler] = pambgfun( ___ ,'Cut','2D')

pambgfun( ___ )
```

Description

`pafmag = pambgfun(X,fs)` returns the magnitude of the normalized periodic ambiguity function (PAF) for one period of the periodic signal `X`. `fs` is the sampling rate.

`[pafmag,delay,doppler] = pambgfun(X,fs)` also returns the time delay vector, `delay`, and the Doppler shift vector, `doppler`. The delay vector is along the zero Doppler cut of the PAF. The Doppler shift vector is along the zero delay cut.

`[pafmag,delay,doppler] = pambgfun(X,fs,P)` returns the magnitude of the normalized PAF for `P` periods of the periodic signal `X`.

`[pafmag,delay] = pambgfun( ___ ,'Cut','Doppler')` returns the PAF, `pafmag`, along a zero Doppler cut. The `delay` argument contains the time delay vector corresponding to the columns of `pafmag`.

`[pafmag,delay] = pambgfun( ___ ,'Cut','Doppler','CutValue',V)` returns the PAF, `pafmag`, along a nonzero Doppler cut specified by `V`. The `delay` argument contains the time delay vector corresponding to the columns of `pafmag`.

`[pafmag,doppler] = pambgfun( ___ ,'Cut','Delay')` returns the PAF, `pafmag`, along the zero delay cut. The `doppler` argument contains the Doppler shift vector corresponding to the rows of `pafmag`.

`[pafmag,doppler] = pambgfun( ___ ,'Cut','Delay','CutValue',V)` returns the PAF, `pafmag`, along a nonzero delay cut specified by `V`. The `doppler` argument contains the Doppler shift vector corresponding to the rows of `pafmag`.

`[pafmag,delay,doppler] = pambgfun( ___ ,'Cut','2D')` returns the PAF, `pafmag`, for all delays and Doppler shifts. The `doppler` argument contains the Doppler shift vector corresponding to the rows of `pafmag`. The `delay` argument contains the time delay vector corresponding to the columns of `pafmag`. You cannot use `'CutValue'` when `'Cut'` is set to `'2D'`.

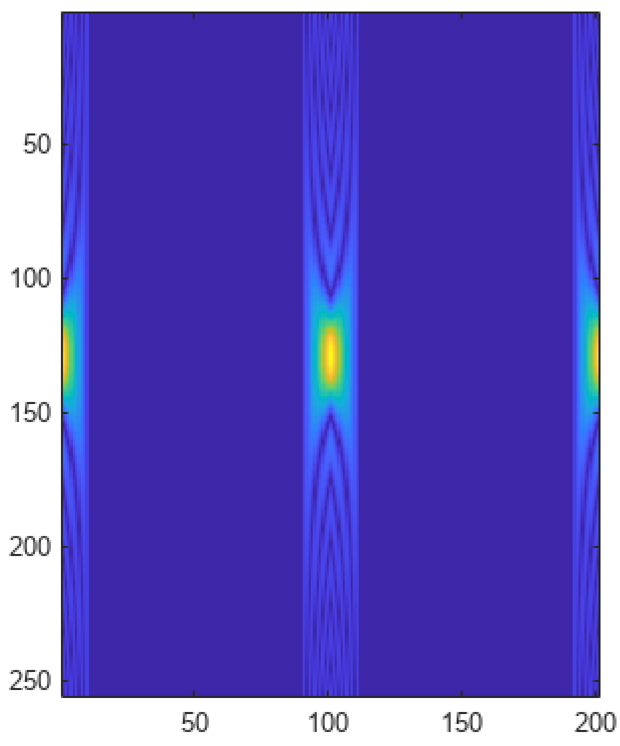
`pambgfun( ___ )` with no output arguments plots the PAF. When `'Cut'` is `'2D'`, the function produces a contour plot of the PAF function. When `'Cut'` is `'Delay'` or `'Doppler'`, the function produces a line plot of the PAF cut.

Examples

Periodic Ambiguity Function for Rectangular Waveform

Plot the PAF function of a rectangular pulse waveform for one period. Assume the pulse repetition frequency (PRF) is 10.0 kHz and that the sampling frequency is a multiple of the PRF.

```
PRF = 10.0e3;
fs = 101*PRF;
waveform = phased.RectangularWaveform('SampleRate',fs,'PulseWidth',1e-5, ...
    'NumPulses',1,'PRF',PRF);
wav = waveform();
pamf = pambgfun(wav,fs);
imagesc(pamf)
axis equal
axis tight
```



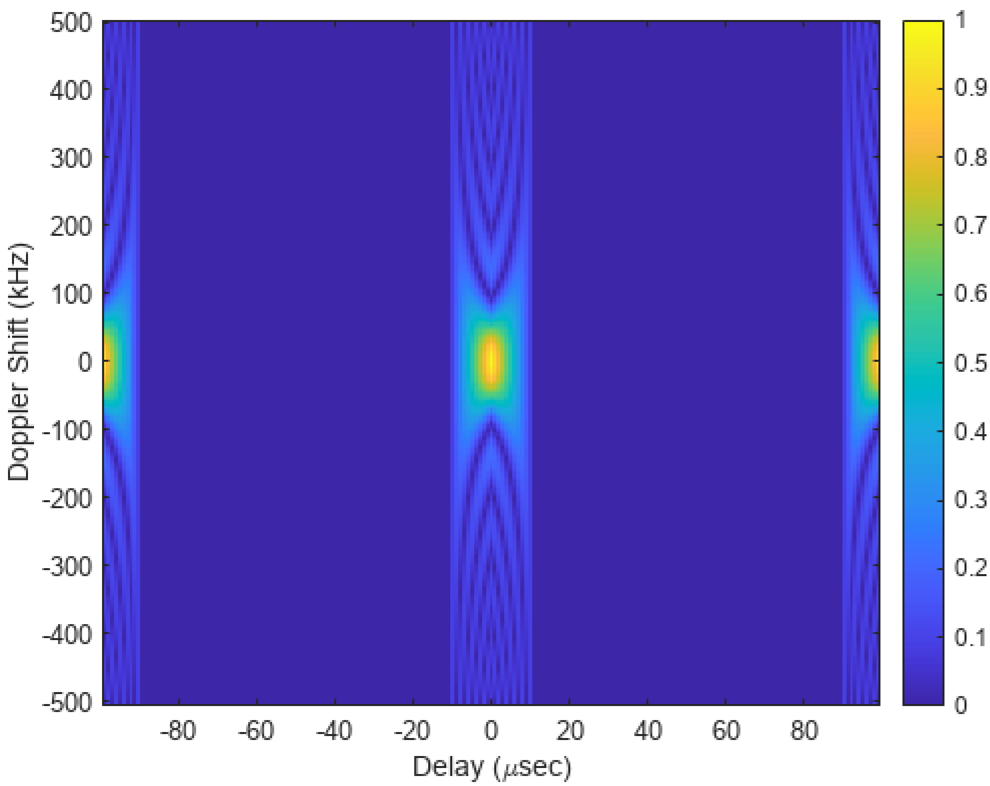
Periodic Ambiguity Function with Delay and Doppler Output

Plot the periodic ambiguity function of a rectangular pulse waveform for one period. Assume the pulse repetition frequency (PRF) is 10.0 kHz and that the sampling frequency is a multiple of the PRF. Return the Doppler and delay values from the `pambgfun` function.

```
PRF = 10.0e3;
fs = 101*PRF;
waveform = phased.RectangularWaveform('SampleRate',fs,'PulseWidth',1e-5, ...
    'NumPulses',1,'PRF',PRF);
wav = waveform();
[pamf,delays,doppler] = pambgfun(wav,fs);
```

Plot the periodic ambiguity function.

```
imagesc(delays*1e6,doppler/1000,pamf)
axis xy
xlabel('Delay (\musec)')
ylabel('Doppler Shift (kHz)')
colorbar
```



Zero Delay Cut of Periodic Ambiguity Function

Plot a cut at zero delay for the periodic ambiguity function of a rectangular pulse waveform for five periods. Assume the pulse repetition frequency is 10.0 kHz and that the sampling frequency is a multiple of the PRF. Return the Doppler and delay values from the function.

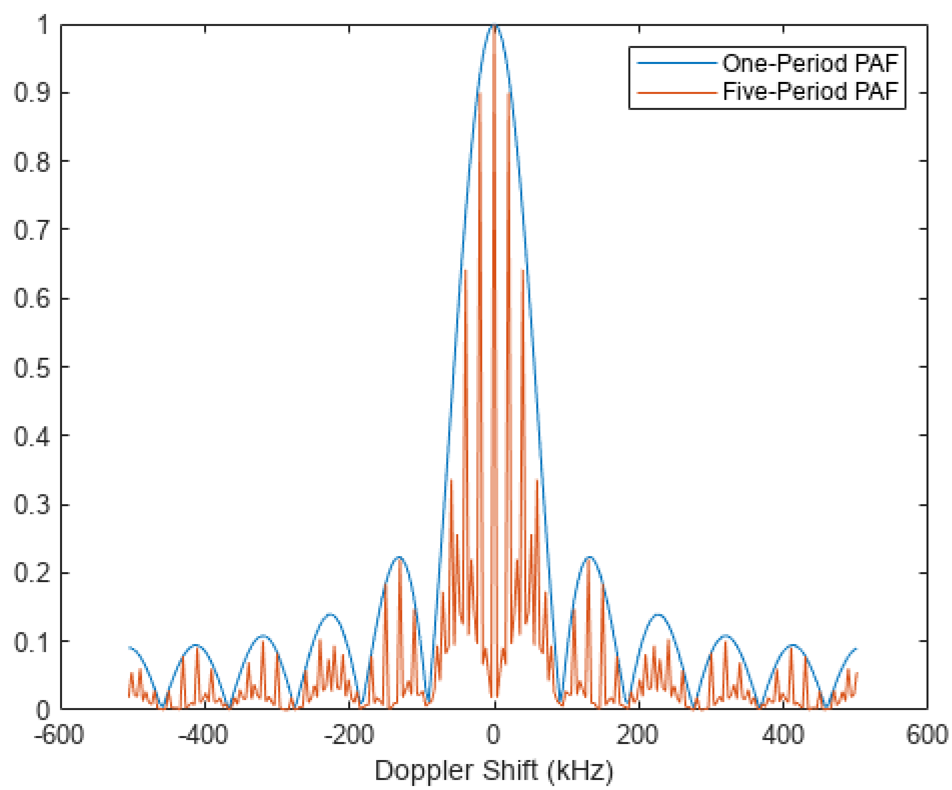
```
PRF = 10.0e3;
fs = 101*PRF;
waveform = phased.RectangularWaveform('SampleRate',fs,'PulseWidth',1e-5, ...
    'NumPulses',1,'PRF',PRF);
wav = waveform();
```

Find the periodic ambiguity functions along a zero delay cut for one and for five periods.

```
[pamf,delays,doppler] = pambgfun(wav,fs,1);
f1 = pamf(:,101);
[pamf,delays,doppler] = pambgfun(wav,fs,5);
f2 = pamf(:,101);
```

Plot the periodic ambiguity functions.

```
plot(doppler/1000,f1)
hold on
plot(doppler/1000,f2)
xlabel('Doppler Shift (kHz)')
legend('One-Period PAF','Five-Period PAF')
```



Zero Doppler Cut of LFM Periodic Ambiguity Function

Plot the zero Doppler cut for the five-period periodic ambiguity function of a linear FM pulse waveform. Assume the pulse repetition frequency (PRF) is 10.0 kHz and that the sampling frequency is a multiple of the PRF. Return the Doppler and delay values from the `pambgfun` function.

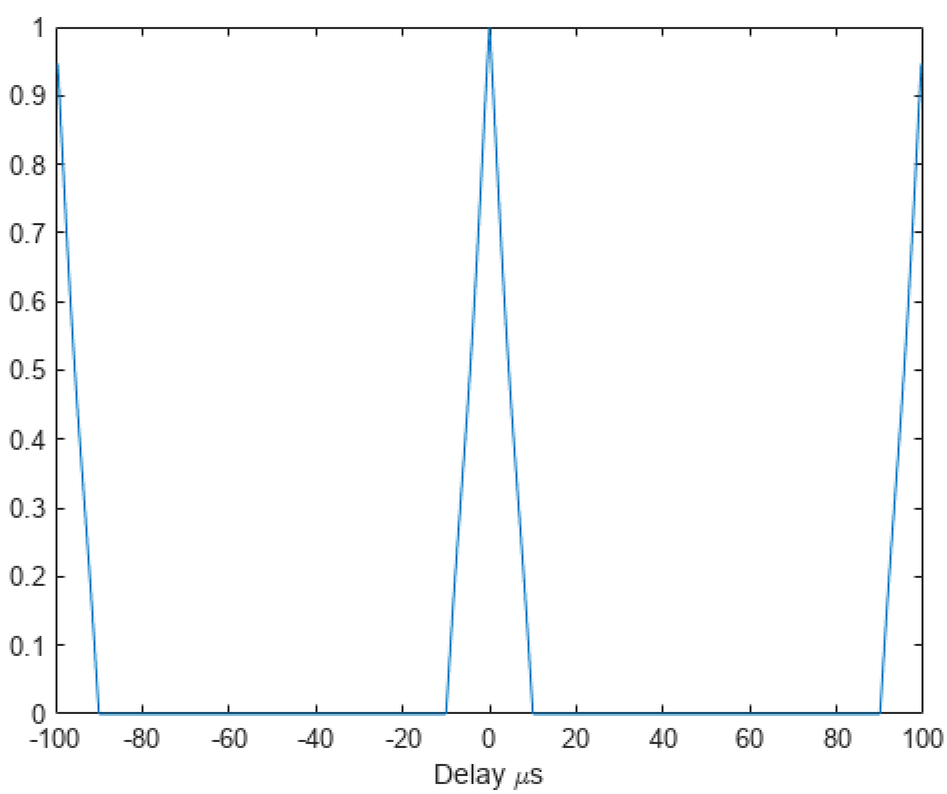
```
PRF = 10.0e3;
fs = 200*PRF;
waveform = phased.LinearFMWaveform('SampleRate',fs,'PulseWidth',1e-5, ...
    'NumPulses',1,'PRF',PRF);
wav = waveform();
```

Find the five-period periodic ambiguity function along a zero Doppler cut.

```
[pamf,delays] = pambgfun(wav,fs,5,'Cut','Doppler');
```

Plot the periodic ambiguity functions.

```
plot(delays*1.0e6,pamf)
xlabel('Delay \mus')
```



Non-Zero Doppler Cut of LFM Periodic Ambiguity Function

Plot a non-zero Doppler cut for the 5-period periodic ambiguity function of a linear FM pulse waveform by explicitly specifying the cut value. Assume the pulse repetition frequency is 10.0e3 Hz and that the sampling frequency is a multiple of the PRF. Return the Doppler and delay values from the function.

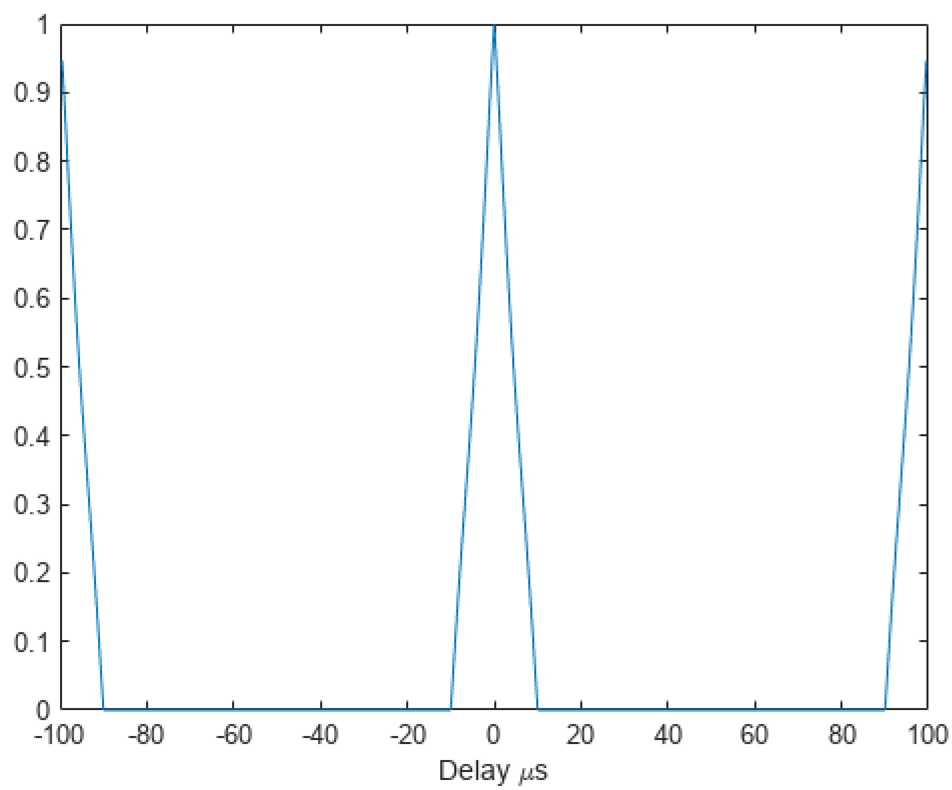
```
PRF = 10.0e3;
fs = 200*PRF;
waveform = phased.LinearFMWaveform('SampleRate',fs,'PulseWidth',1e-5,...
    'NumPulses',1,'PRF',PRF);
wav = waveform();
```

Find the 5-period periodic ambiguity function along a non-zero Doppler cut.

```
dopval = 20.0;
[pamf,delays] = pambgfun(wav,fs,5,'Cut','Doppler','CutValue',dopval);
```

Plot the periodic ambiguity functions.

```
plot(delays*1.0e6,pamf)
xlabel('Delay \mus')
```



Zero-Delay Cut of FMCW Periodic Ambiguity Function

Plot a zero delay cut for the three-period periodic ambiguity function of an FMCW waveform. Assume a sweep bandwidth of 100 kHz with a sampling frequency of 1 MHz. Return and plot the Doppler shift values.

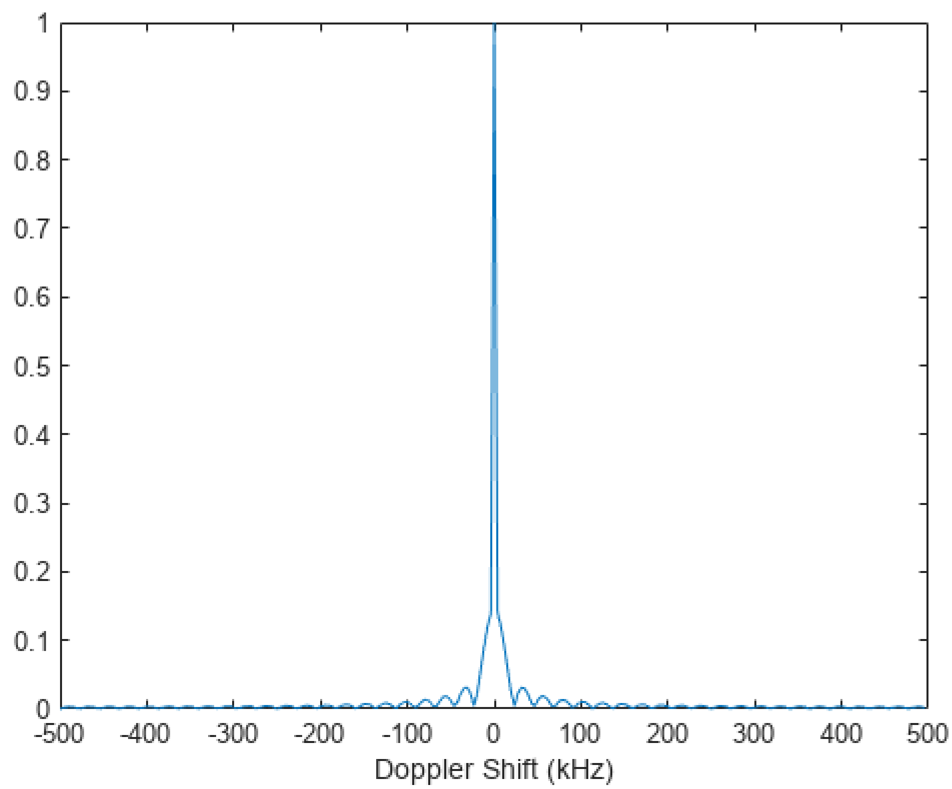
```
fs = 1.0e6;
waveform = phased.FMCWWaveform('SweepBandwidth',100.0e3,'SampleRate',fs, ...
    'OutputFormat','Sweeps','NumSweeps',1);
wav = waveform();
```

Find the three-period periodic ambiguity function along a zero delay cut.

```
[pamf,doppler] = pambgfun(wav,fs,3,'Cut','Delay');
```

Plot the zero delay cut of the periodic ambiguity function.

```
plot(doppler/1.0e3,pamf)
xlabel('Doppler Shift (kHz)')
```



Nonzero Delay Cut of FMCW Periodic Ambiguity Function

Plot a non-zero delay cut of -20 µs for the three-period periodic ambiguity function of an FMCW waveform. Assume a sweep bandwidth of 100 kHz with a sampling frequency of 1 MHz. Return and plot the Doppler shift values.

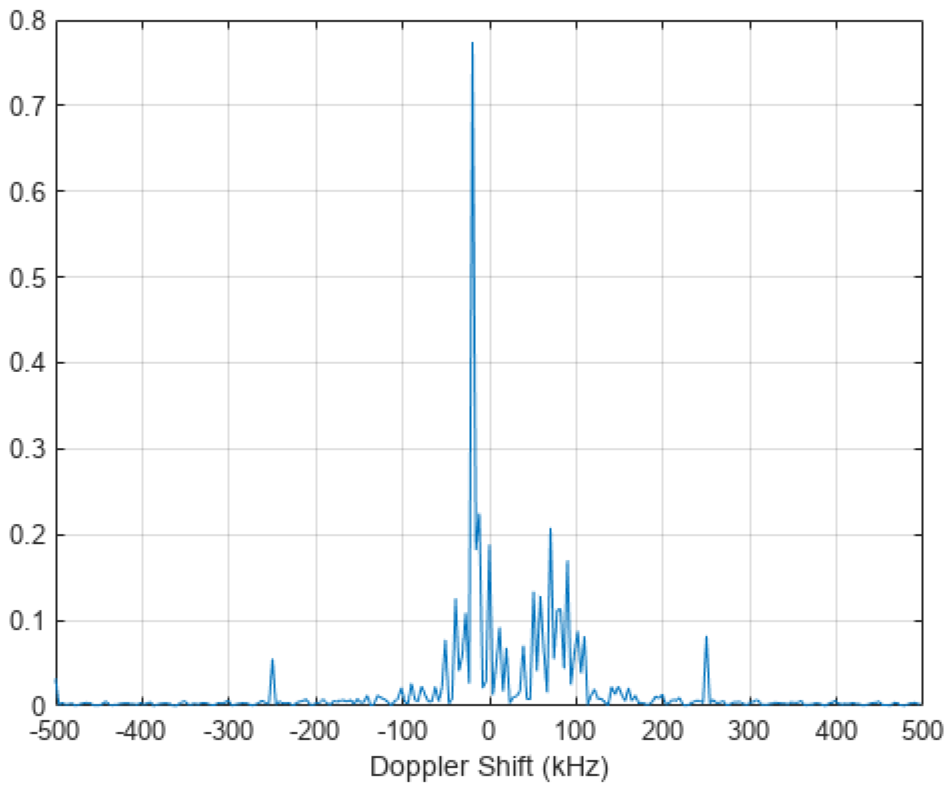
```
fs = 1.0e6;
waveform = phased.FMCWWaveform('SweepBandwidth',100.0e3,'SampleRate',fs, ...
    'OutputFormat','Sweeps','NumSweeps',1,'SweepTime',100e-6);
wav = waveform();
```

Find the three-period periodic ambiguity function along a nonzero Delay cut.

```
delayval = -20.0e-6;
[pamf,doppler] = pambgfun(wav,fs,3,'Cut','Delay','CutValue',delayval);
```

Plot the nonzero delay cut of the periodic ambiguity function.

```
plot(doppler/1.0e3,pamf)
grid
xlabel('Doppler Shift (kHz)')
```



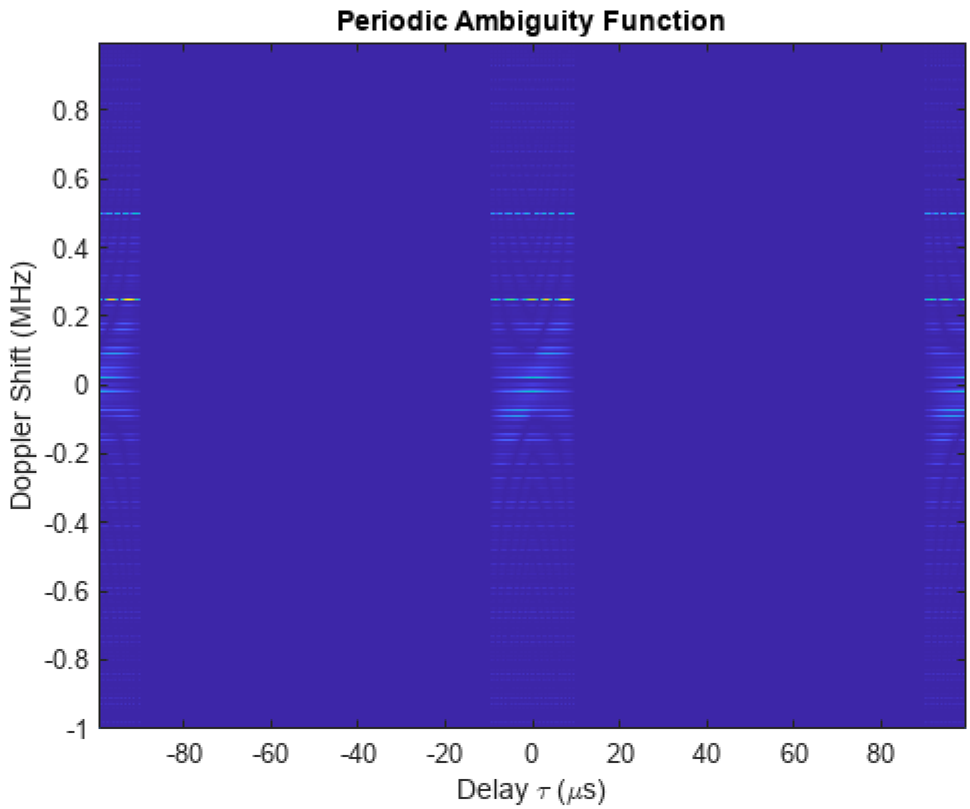
Linear FM Periodic Ambiguity Diagram Image

Display an image of the 9-period periodic ambiguity function for a linear FM pulse waveform. Assume the pulse repetition frequency is 10.0e3 Hz and that the sampling frequency is a multiple of the PRF.

```
PRF = 10.0e3;
fs = 200*PRF;
waveform = phased.LinearFMWaveform('SampleRate',fs,'PulseWidth',1e-5,...
    'NumPulses',1,'PRF',PRF);
wav = waveform();
```

Compute and display the 9-period periodic ambiguity function for all delays and frequencies.

```
[pamf,delays,doppler] = pambgfun(wav,fs,9,'Cut','2D');
imagesc(delays*1e6,doppler/1e6,pamf)
title('Periodic Ambiguity Function')
xlabel('Delay \tau ({\mu}s)')
ylabel('Doppler Shift (MHz)')
axis xy
```



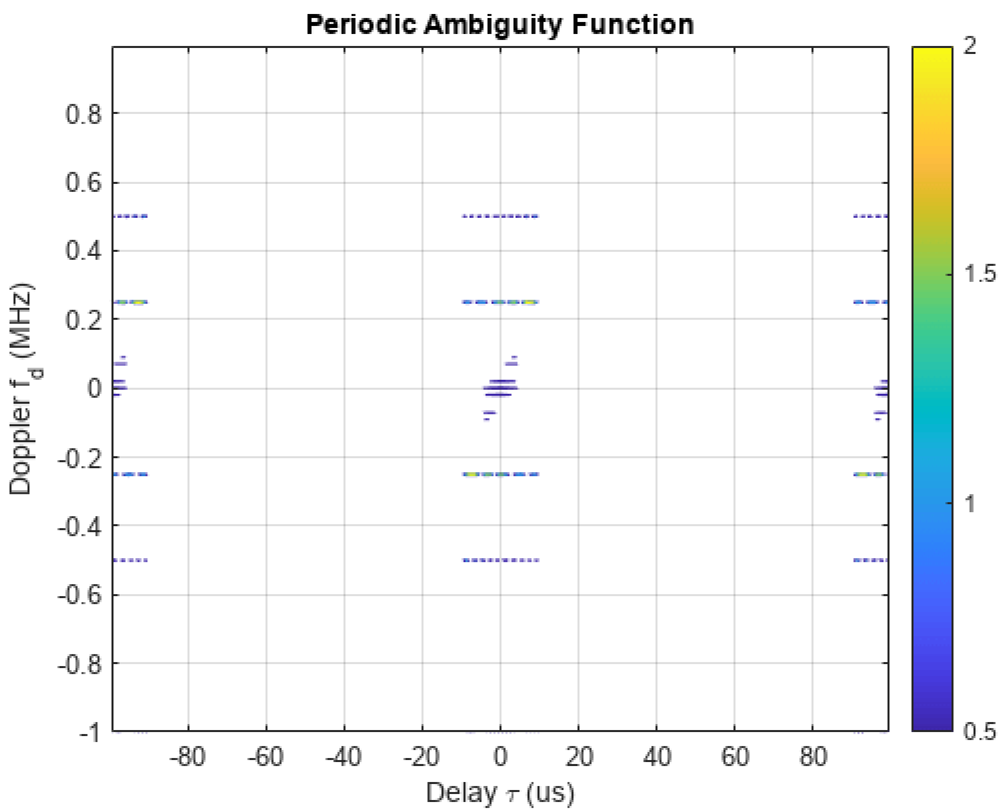
Linear FM Periodic Ambiguity Diagram

Plot the seven-period periodic ambiguity function of a linear FM pulse waveform. Assume the pulse repetition frequency (PRF) is 10.0 kHz and that the sampling frequency is a multiple of the PRF.

```
PRF = 10.0e3;
fs = 200*PRF;
waveform = phased.LinearFMWaveform('SampleRate',fs,'PulseWidth',1e-5, ...
    'NumPulses',1,'PRF',PRF);
wav = waveform();
```

Find the periodic ambiguity function.

```
pambgfun(wav,fs,7,'Cut','2D')
```



Input Arguments

X — Input pulse waveform
complex-valued vector

Input pulse waveform, specified as a complex-valued vector.

Example: `[0,.1,.3,.4,.3,.1.0]`

Data Types: `double`
Complex Number Support: Yes

`fs` — Sampling frequency
positive scalar

Sampling frequency, specified as a positive scalar. Units are in hertz.

Example: `3e3`

Data Types: `double`

P — Number of periods
`1` (default) | positive integer

Number of periods, specified as a positive integer.

Example: `5`

Data Types: `double`

V — Optional time delay or Doppler shift at which ambiguity function cut is taken
0 (default) | real-valued scalar

When you set `'Cut'` to `'Delay'` or `'Doppler'`, use `V` to specify a nonzero cut value. You cannot use `V` when you set `'Cut'` to `'2D'`.

When `'Cut'` is set to `'Delay'`, `V` is the time delay at which the cut is taken. Time delay units are in seconds.

When `'Cut'` is set to `'Doppler'`, `V` is the Doppler frequency shift at which the cut is taken. Doppler units are in hertz.

Example: `10.0`

Data Types: `double`

Output Arguments

pafmag — Normalized PAF function magnitude
real-valued *M*-by-*N* matrix | real-valued *M*-element column vector | real-valued *N*-element row vector

Normalized PAF function magnitude, returned as a vector or a matrix of nonnegative real values. The dimensions of `pafmag` depend on the value of `'Cut'`.

'Cut'	pafmagdimensions
'2D'	*M*-by-*N* matrix.
'Delay'	*M*-element column vector.
'Doppler'	*N*-element row vector.

M is the number of Doppler frequencies and *N* is the number of time delays.

delay — Time delay vector
real-valued *N*-element vector

Time delay vector, returned as an *N*-by-1 vector. If *N* is the length of signal `X`, then the delay vector consist of $2N - 1$ samples in the range, $-(N/2) - 1,\ldots,(N/2) - 1)$.

doppler — Doppler shift vector
real-valued *M* vector

Doppler shift vector, returned as an *M*-by-1 vector of Doppler frequencies. The Doppler frequency vector consists of $M = 2^{ceil(log2\ N)}$ equally-spaced samples. Frequencies are $(-(M/2)F_s,\ldots,(M/2-1)F_s)$.

More About

Periodic Ambiguity Function

The periodic ambiguity function (PAF) is an extension of the ordinary ambiguity function to periodic waveforms.

Use this function analyze the response of a correlation receiver to a time-delayed or Doppler-shifted narrowband periodic waveform. Narrowband periodic signals consist of CW tones modulated by a periodic complex envelope. These types of signals are commonly used in radar systems to form transmitted pulse trains.

A time periodic waveform has the property *y(t + T) = y(t)*, where *T* is the period. The PAF function for an *N*-period waveform is defined as

$$A_{NT}(\tau,\nu) = \frac{1}{NT}\int_0^{NT} y(t+\frac{\tau}{2})y^*(t-\frac{\tau}{2})e^{i2\pi\nu t}dt$$

Taking advantage of the periodicity, you can rewrite the function as

$$A_{NT}(\tau,\nu) = \frac{1}{NT}\sum_{n=1}^{N} e^{i2\pi\nu(n-1)T}\int_0^{T} u(t+\frac{\tau}{2})u^*(t-\frac{\tau}{2})e^{i2\pi\nu s}ds$$

The last term on the right side is the one-period PAF function, $A_T(\tau,\nu)$. The first term on the right side is due to Doppler only. The Doppler term is proportional to the periodic *sinc()* function and you can rewrite the periodic ambiguity function as

$$A_{NT}(\tau,\nu) = \frac{\sin 2\pi\nu NT}{N\sin 2\pi\nu T}e^{i2\pi\nu(N-1)T}A_T(\tau,\nu)$$

The Doppler term improves the Doppler resolution by a factor of *1/NT*.

The one-period PAF function is not the same as the ordinary ambiguity because the integration limits are different.

Version History

Introduced in R2016b

References

[1] Levanon, N. and E. Mozeson. *Radar Signals*. Hoboken, NJ: John Wiley & Sons, 2004.

[2] Mahafza, B. R., and A. Z. Elsherbeni. *MATLAB Simulations for Radar Systems Design*. Boca Raton, FL: CRC Press, 2004.

[3] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.
- Supported only when output arguments are specified.

See Also

Functions
ambgfun | dutycycle

Objects
phased.LinearFMWaveform | phased.MatchedFilter | phased.PhaseCodedWaveform | phased.RectangularWaveform | phased.SteppedFMWaveform

phitheta2azel

Convert angles from phi/theta form to azimuth/elevation form

Syntax

```
AzEl = phitheta2azel(PhiTheta)
AzEl = phitheta2azel(PhiTheta,RotAx)
```

Description

`AzEl = phitheta2azel(PhiTheta)` converts the phi/theta angle on page 2-261 pairs to their corresponding azimuth/elevation angle on page 2-260 pairs.

`AzEl = phitheta2azel(PhiTheta,RotAx)` also specifies the choice of phi-theta angle convention using `RotAx`.

Examples

Convert Phi-Theta Coordinates to Azimuth-Elevation Coordinates

Find the azimuth-elevation representation for φ = 30° and θ = 0°. Use the phi-theta convention with φ defined from the *y*-axis to the *z*-axis, and θ defined from the *x*-axis toward the *yz*-plane.

```
azel = phitheta2azel([30;10])
```

```
azel = 2×1

    8.6822
    4.9809
```

Rotate and Convert Phi-Theta Coordinates to Azimuth-Elevation Coordinates

Find the azimuth-elevation representation for φ = 30° and θ = 0°. Use the phi-theta convention with φ defined from the x-axis to the y-axis, and θ defined from the z-axis toward the xy-plane.

```
azel = phitheta2azel([30;10],false)
```

```
azel = 2×1

    30
    80
```

Input Arguments

`PhiTheta` — Phi-theta angle pairs
two-row matrix

Phi and theta angles, specified as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [phi; theta].

Data Types: `double`

`RotAx` — Phi-theta angle convention selection
`true` (default) | `false`

Phi-theta angle convention selection, specified as `true` or `false`.

- If `RotAx` is `true`, the phi angle of a direction vector is the angle from the *z*-axis to the projection of the vector into the *yz*-plane. The theta angle is defined from the *x*-axis to the direction vector. Positive values are toward the *yz*-plane.
- If `RotAx` is `false`, the phi angle is defined from the *x*-axis to the projection of the direction vector in the *xy*-plane. The angle is positive in the direction of the *y*-axis. The theta angle is defined from the *z*-axis to the direction vector and is positive in the direction of the *xy*- plane (see “Alternative Definition of Phi and Theta Angles” on page 2-262).

Data Types: `logical`

Output Arguments

`AzEl` — Azimuth-elevation angle pairs
two-row matrix

Azimuth and elevation angles, returned as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [azimuth; elevation]. The matrix dimensions of `AzEl` are the same as those of `PhiTheta`.

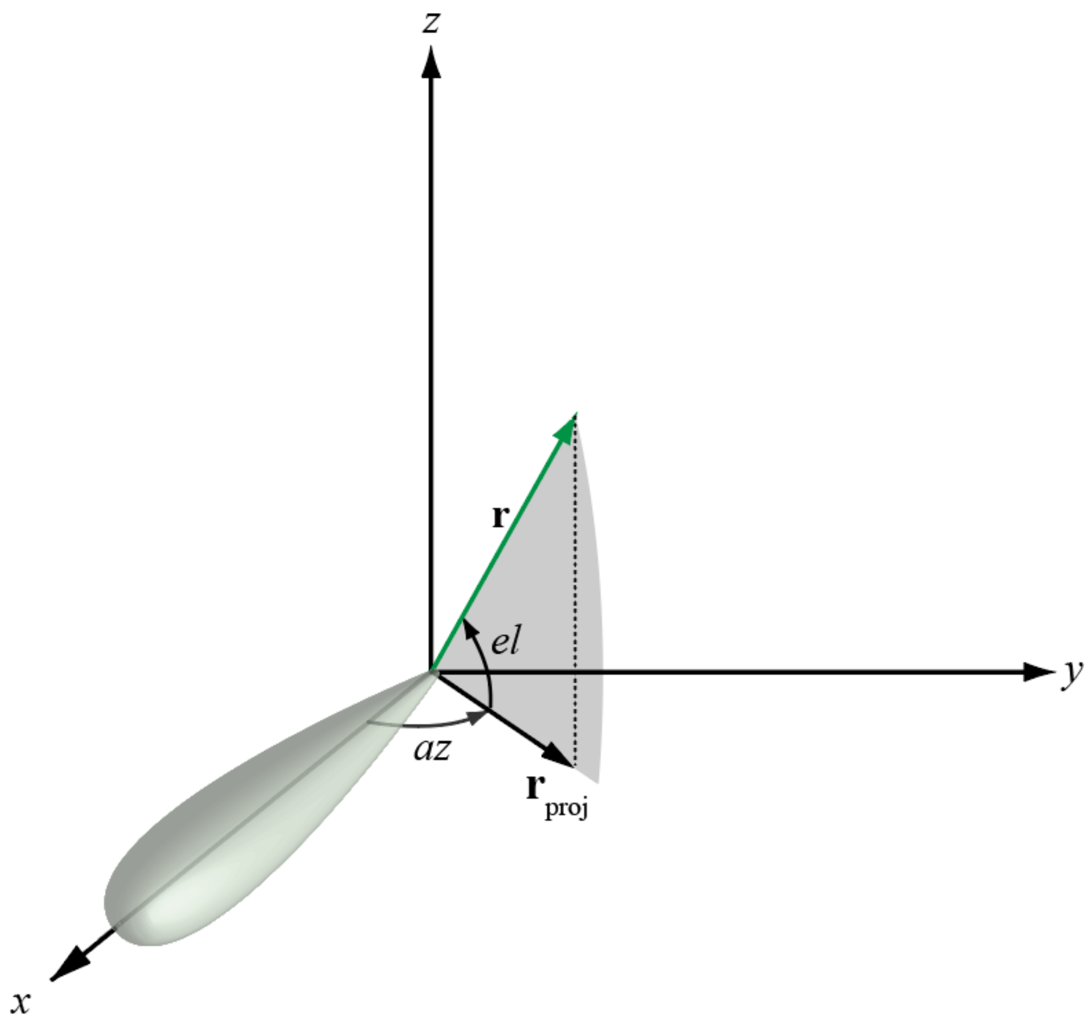
More About

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

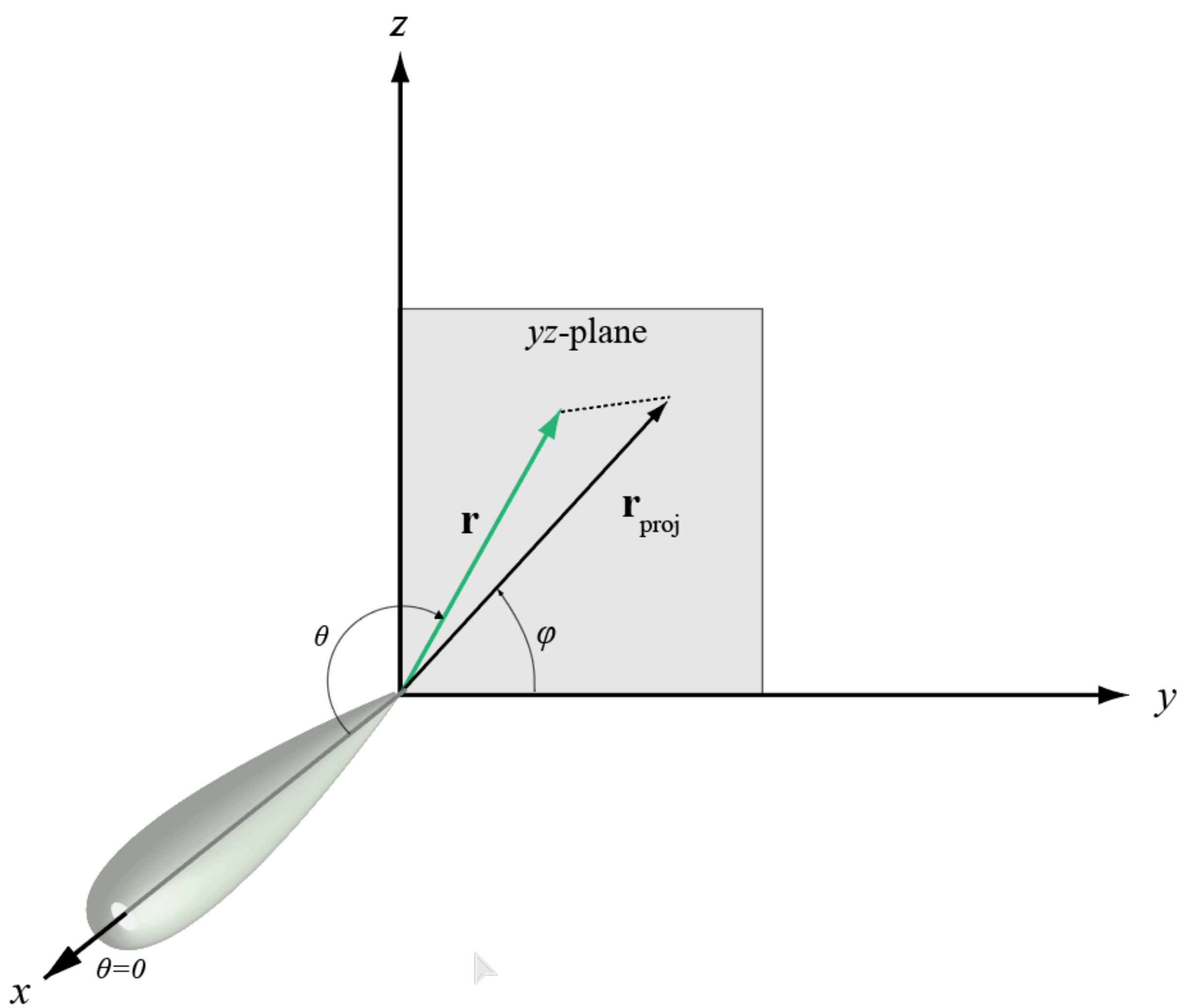
This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



Phi and Theta Angles

The phi angle (φ) is the angle from the positive *y*-axis to the vector’s orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



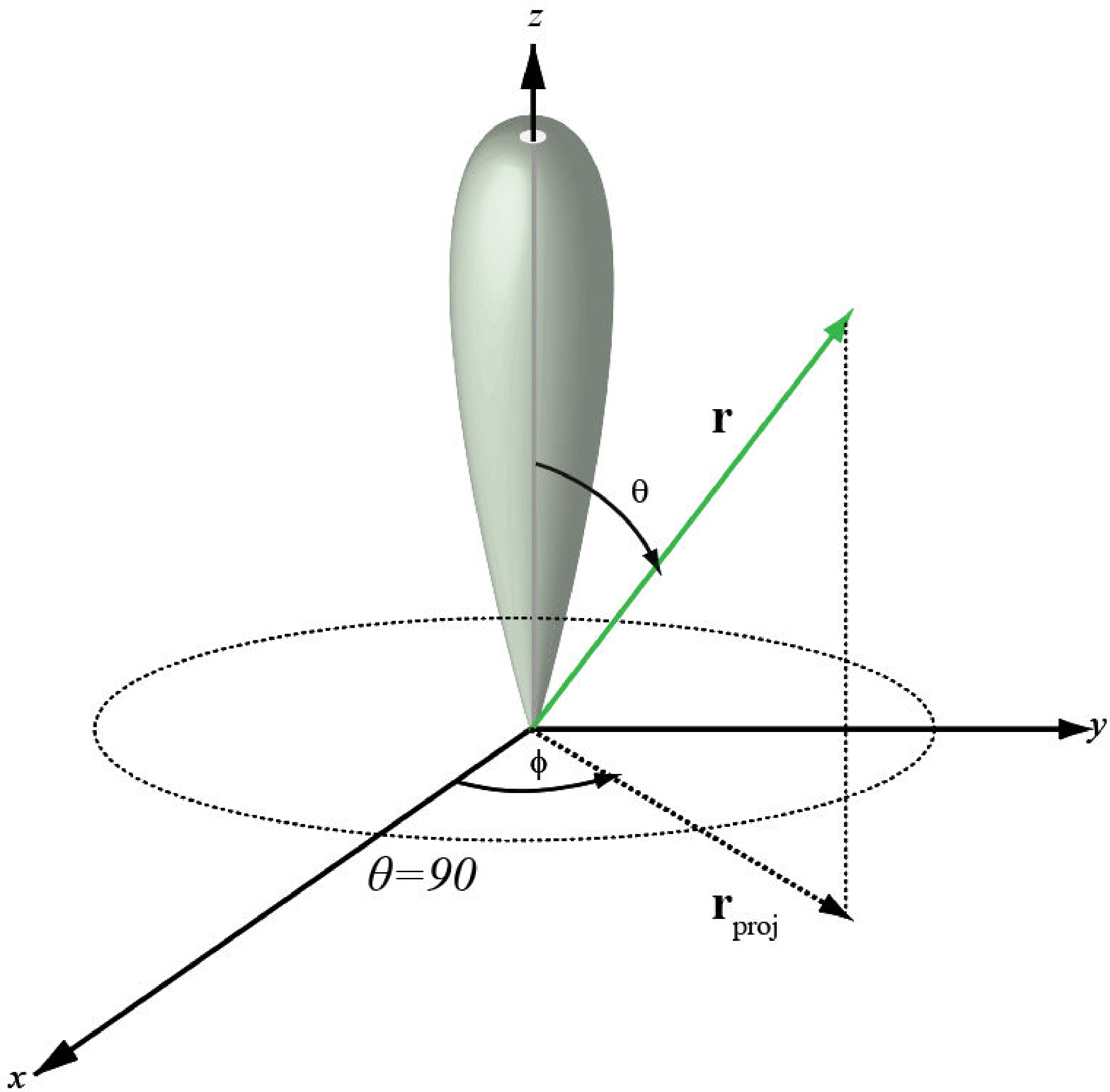
The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi\sin\theta$
$\tan az = \cos\phi\tan\theta$
$\cos\theta = \cos el\cos az$
$\tan\phi = \tan el/\sin az$

Alternative Definition of Phi and Theta Angles

The phi angle (φ) is the angle from the positive x-axis to the vector's orthogonal projection onto the xy plane. The angle is positive toward the positive y-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the z-axis to the vector itself. The angle is positive toward the xy plane. The theta angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line.



$\phi = az$
$\theta = 90 - el$
$az = \phi$
$el = 90 - \theta$

This transformation applies when `RotAx` is `false`.

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`azel2phitheta`

Topics
“Spherical Coordinates”

phitheta2azelpat

Convert radiation pattern from phi-theta coordinates to azimuth-elevation coordinates

Syntax

```
pat_azel = phitheta2azelpat(pat_phitheta,phi,theta)
pat_azel = phitheta2azelpat(pat_phitheta,phi,theta,az,el)
pat_azel = phitheta2azelpat( ___ ,'RotateZ2X',rotpatax)
[pat_azel,az_pat,el_pat] = phitheta2azelpat( ___ )
```

Description

`pat_azel = phitheta2azelpat(pat_phitheta,phi,theta)` converts the antenna radiation pattern, `pat_phitheta`, from phi and theta coordinates to the pattern `pat_azel` in azimuth and elevation coordinates. `phi` and `theta` are the phi and theta coordinates at which `pat_phitheta` values are defined. The `pat_azel` matrix covers azimuth values from –180 to 180 degrees and elevation values from –90 to 90 degrees in one degree increments. The function interpolates the `pat_phitheta` matrix to estimate the response of the antenna in a given direction.

`pat_azel = phitheta2azelpat(pat_phitheta,phi,theta,az,el)` uses vectors `az` and `el` to specify the grid at which to sample `pat_azel`. To avoid interpolation errors, `az` should cover the range [–180, 180] and `el` should cover the range [–90, 90].

`pat_azel = phitheta2azelpat( ___ ,'RotateZ2X',rotpatax)` also specifies `rotpatax` to indicate the boresight direction of the pattern: *x*-axis or *z*-axis.

`[pat_azel,az_pat,el_pat] = phitheta2azelpat( ___ )` also returns vectors `az_pat` and `el_pat` containing the azimuth and elevation angles at which `pat_azel` is sampled.

Examples

Convert Radiation Pattern to Azimuth and Elevation

Convert a radiation pattern to azimuth/elevation form, with the azimuth and elevation angles spaced 1° apart.

Define the pattern in terms of φ and θ.

```
phi = 0:360;
theta = 0:180;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Convert the pattern to azimuth/elevation space.

```
pat_azel = phitheta2azelpat(pat_phitheta,phi,theta);
```

Plot Converted Radiation Pattern

Convert a radiation pattern from theta/phi coordinates to azimuth/elevation coordinates, with azimuth and elevation angles spaced 1° apart.

Define the pattern in terms of phi, ϕ, and theta, θ, coordinates.

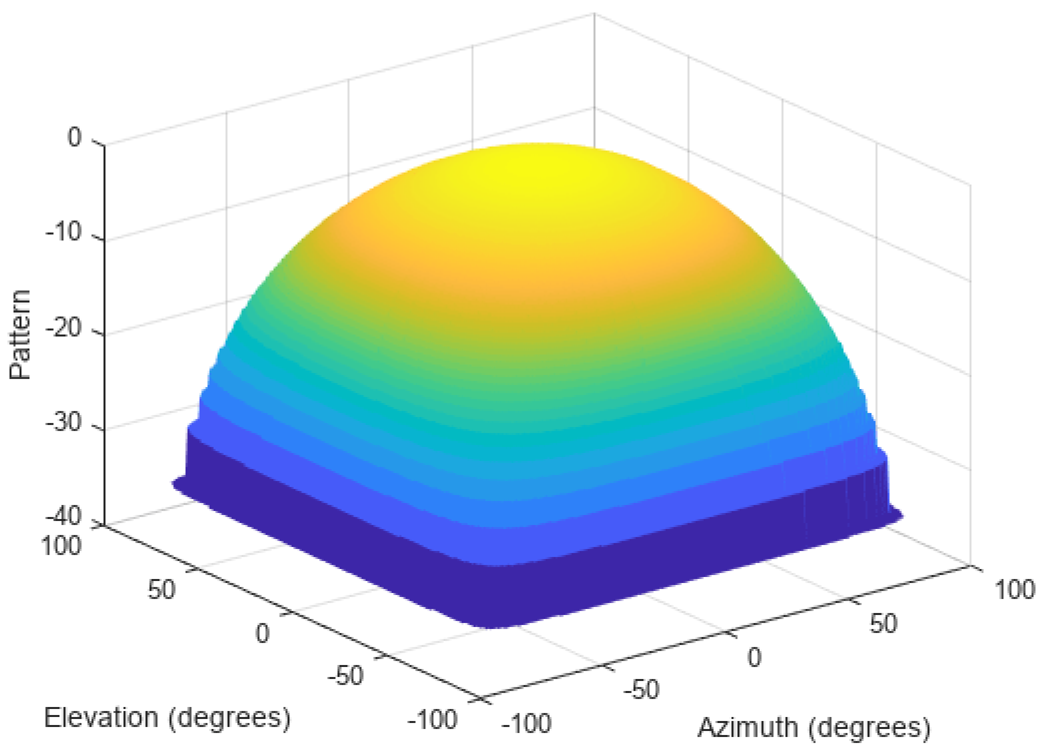
```
phi = 0:360;
theta = 0:180;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Convert the pattern to azimuth/elevation coordinates. Get the azimuth and elevation angles for use in plotting.

```
[pat_azel,az,el] = phitheta2azelpat(pat_phitheta,phi,theta);
```

Plot the radiation pattern.

```
H = surf(az,el,pat_azel);
H.LineStyle = 'none';
xlabel('Azimuth (degrees)');
ylabel('Elevation (degrees)');
zlabel('Pattern');
```

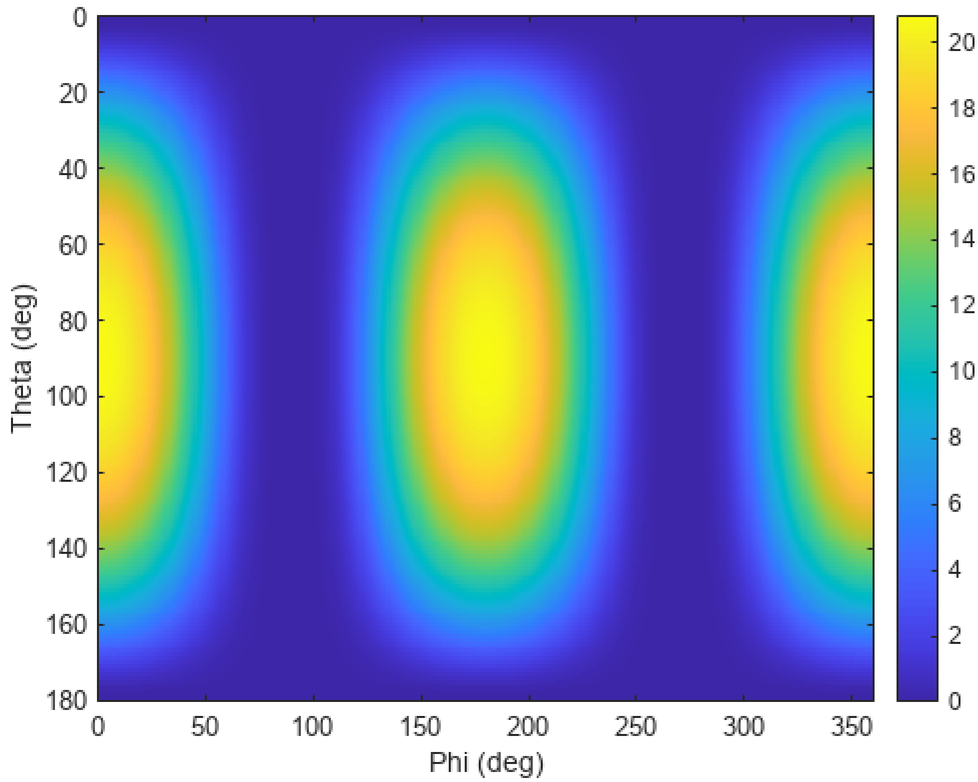


Convert Radiation Pattern from Alternate Phi-Theta Coordinates to Azimuth and Elevation

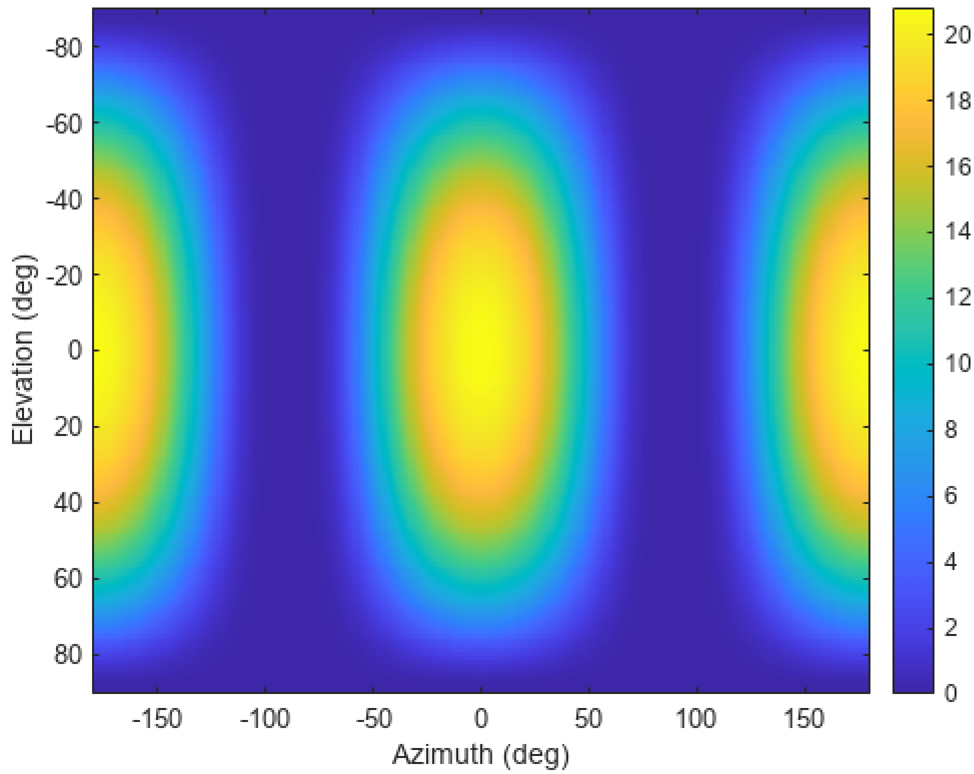
Convert a radiation pattern to the azimuth-elevation coordinates from alternative phi-theta coordinates, with the phi and theta angles spaced one degree apart.

Create a simple radiation pattern in terms of phi and theta. Add an offset to the pattern to suppress taking the logarithm of zero in mag2db.

```
phi = 0:360;
theta = 0:180;
pat_phitheta = mag2db(10*sind(theta').^2*cosd(phi).^4 + 1);
imagesc(phi,theta,pat_phitheta)
xlabel('Phi (deg)')
ylabel('Theta (deg)')
colorbar
```




```
[pat_azel,az_pat,el_pat] = phitheta2azelpat(pat_phitheta,phi,theta,'RotateZ2X',false);
imagesc(az_pat,el_pat,pat_azel)
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
colorbar
```



Convert Radiation Pattern for Specific Azimuth/Elevation Values

Convert a radiation pattern from phi/theta coordinates to azimuth/elevation coordinates, with the azimuth and elevation angles spaced 5˚ apart.

Define the pattern in terms of phi and theta.

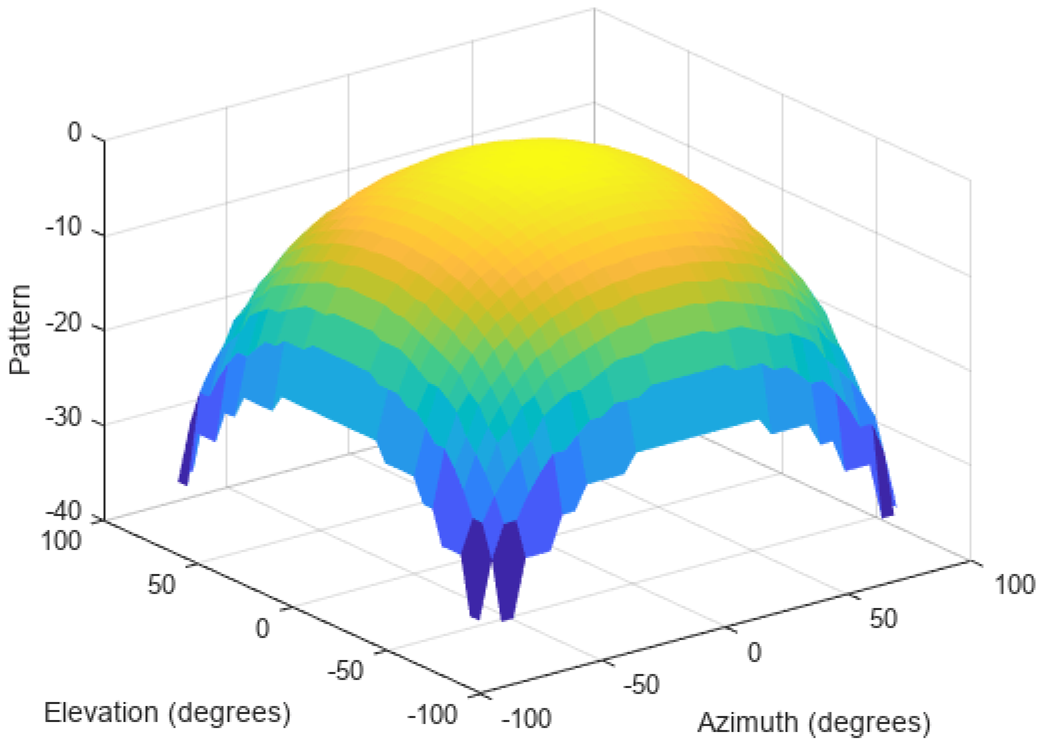
```
phi = 0:360;
theta = 0:180;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Define the set of azimuth and elevation angles at which to sample the pattern. Then, convert the pattern.

```
az = -180:5:180;
el = -90:5:90;
pat_azel = phitheta2azelpat(pat_phitheta,phi,theta,az,el);
```

Plot the radiation pattern.

```
H = surf(az,el,pat_azel);
H.LineStyle = 'none';
xlabel('Azimuth (degrees)');
ylabel('Elevation (degrees)');
zlabel('Pattern');
```



Input Arguments

pat_phitheta — Antenna radiation patter
real-valued *Q*-by-*P* matrix

Antenna radiation pattern in phi-theta coordinates, specified as a real-valued *Q*-by-*P* matrix. `pat_phitheta` contains the magnitude pattern. *P* is the length of the `phi` vector, and *Q* is the length of the `theta` vector. Units are in dB.

Data Types: `double`

phi — Phi angles
real-valued length-*P* vector

Phi angles at which `pat_phitheta` is sampled, specified as a vector of real-valued length-*P* vector. Phi angles lie between 0 and 360, inclusive. Units are in degrees.

Data Types: `double`

theta — Theta angles
real-valued length-*Q* vector

Theta angles at which `pat_phitheta` is sampled, specified as a vector of real-valued length-*Q* vector. Theta angles lie between 0 and 180, inclusive. Units are in degrees.

Data Types: `double`

az — Azimuth angles
[-180:180] (default) | real-valued length-*L* vector

Azimuth angles at which pat_azel samples the pattern, specified as a vector of real-valued length-*L* vector. Azimuth angles lie between –180 and 180, inclusive. Units are in degrees.

Data Types: double

el — Elevation angles
[-90:90] (default) | real-valued length-*M* vector

Elevation angles at which pat_azel samples the pattern, specified as a real-valued length-*M* vector. Elevation angle lie between –90 and 90, inclusive. Units are in degrees.

Data Types: double

rotpatax — Pattern boresight direction selector
true (default) | false

Pattern boresight direction selector, specified as true or false.

- If rotpatax is true, the pattern boresight is along the *x*-axis. In this case, the *z*-axis of phi-theta space is aligned with the *x*-axis of azimuth and elevation space. The phi angle is defined from the *y*-axis to the *z*-axis and the theta angle is defined from the *x*-axis toward the *yz*-plane. (See "Phi and Theta Angles" on page 2-272).
- If rotpatax is false, the phi angle is defined from the *x*-axis to the *y*-axis and the theta angle is defined from the *z*-axis toward the *xy*-plane. (See "Alternative Definition of Phi and Theta" on page 2-273).

Data Types: logical

Output Arguments

pat_azel — Antenna radiation pattern in azimuth-elevation coordinates
real-valued *M*-by-*L* matrix

Antenna radiation pattern in azimuth-elevation coordinates, returned as a real-valued *M*-by-*L* matrix. pat_azel represents the magnitude pattern. *L* is the length of the az_pat vector, and *M* is the length of the el_pat vector. Units are in dB.

az_pat — Azimuth angles
real-valued length-*L* vector

Azimuth angles at which the pat_azel output pattern is sampled, returned as a real-valued length-*L* vector. Units are in degrees.

el_pat — Elevation angles
real-valued length-*M* vector

Elevation angles at which the pat_azel output pattern is sampled, returned as a real-valued length-*M* vector. Units are in degrees.

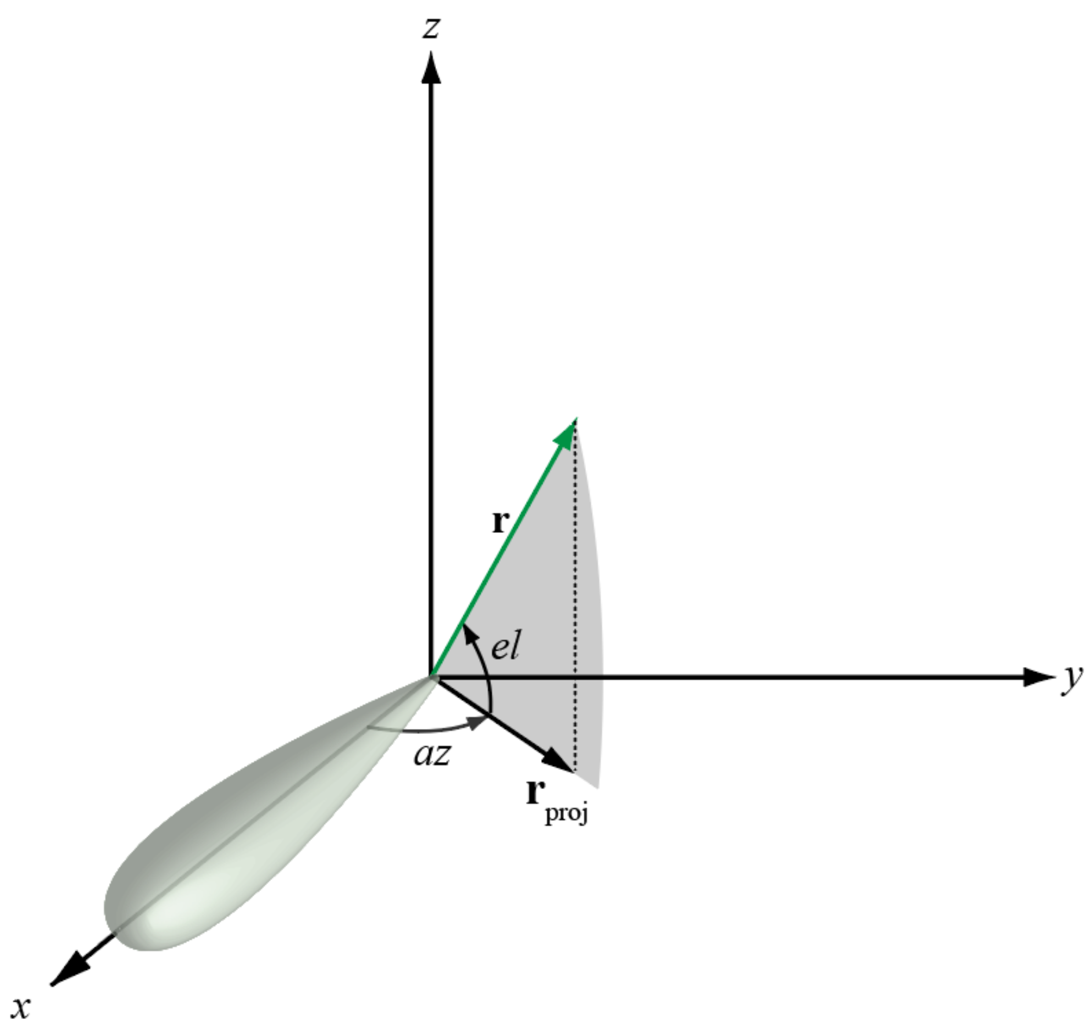
More About

Azimuth and Elevation Angles

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

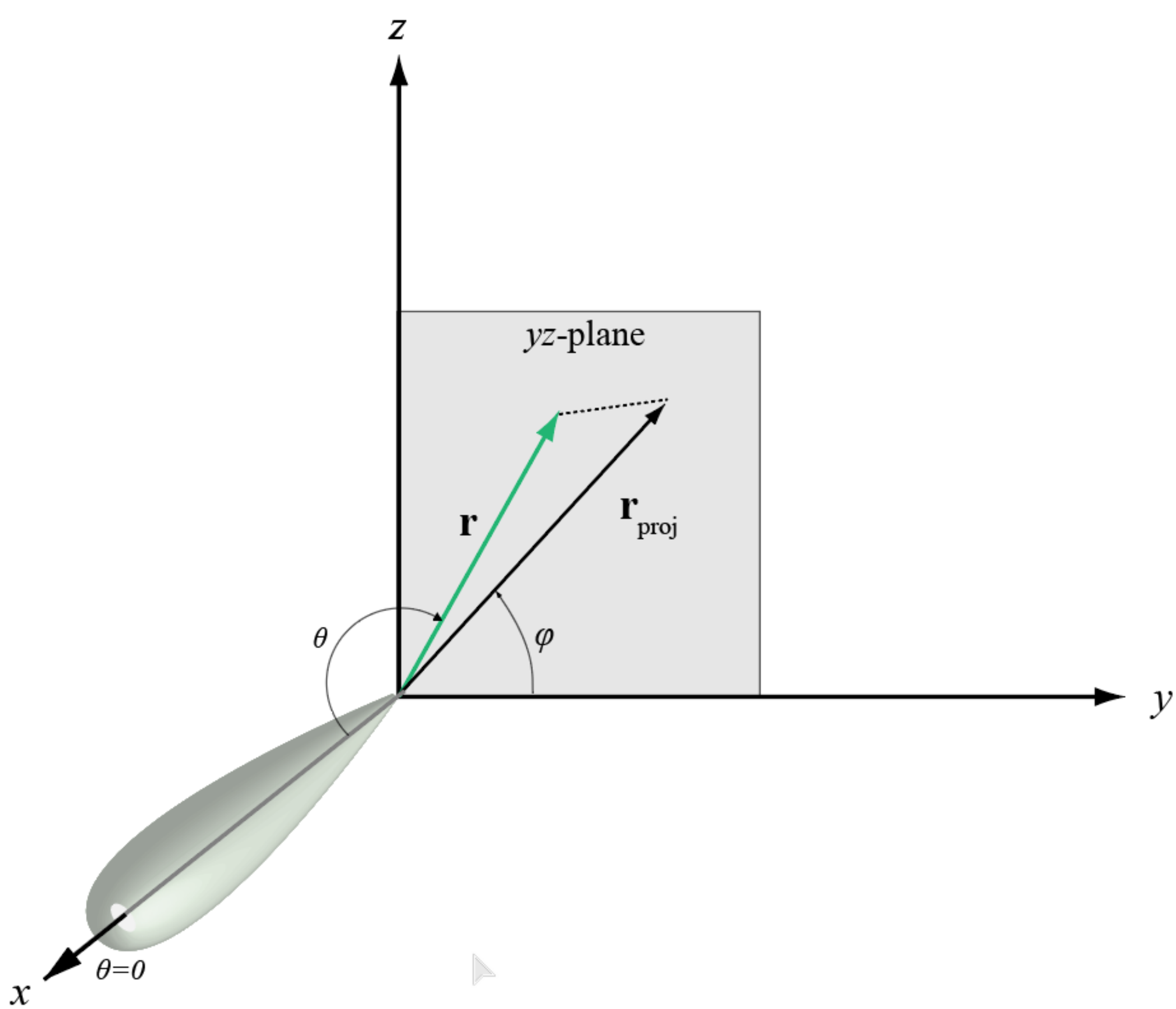
This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



Phi and Theta Angles

The phi angle (φ) is the angle from the positive *y*-axis to the vector's orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi \sin\theta$

$\tan az = \cos\phi \tan\theta$

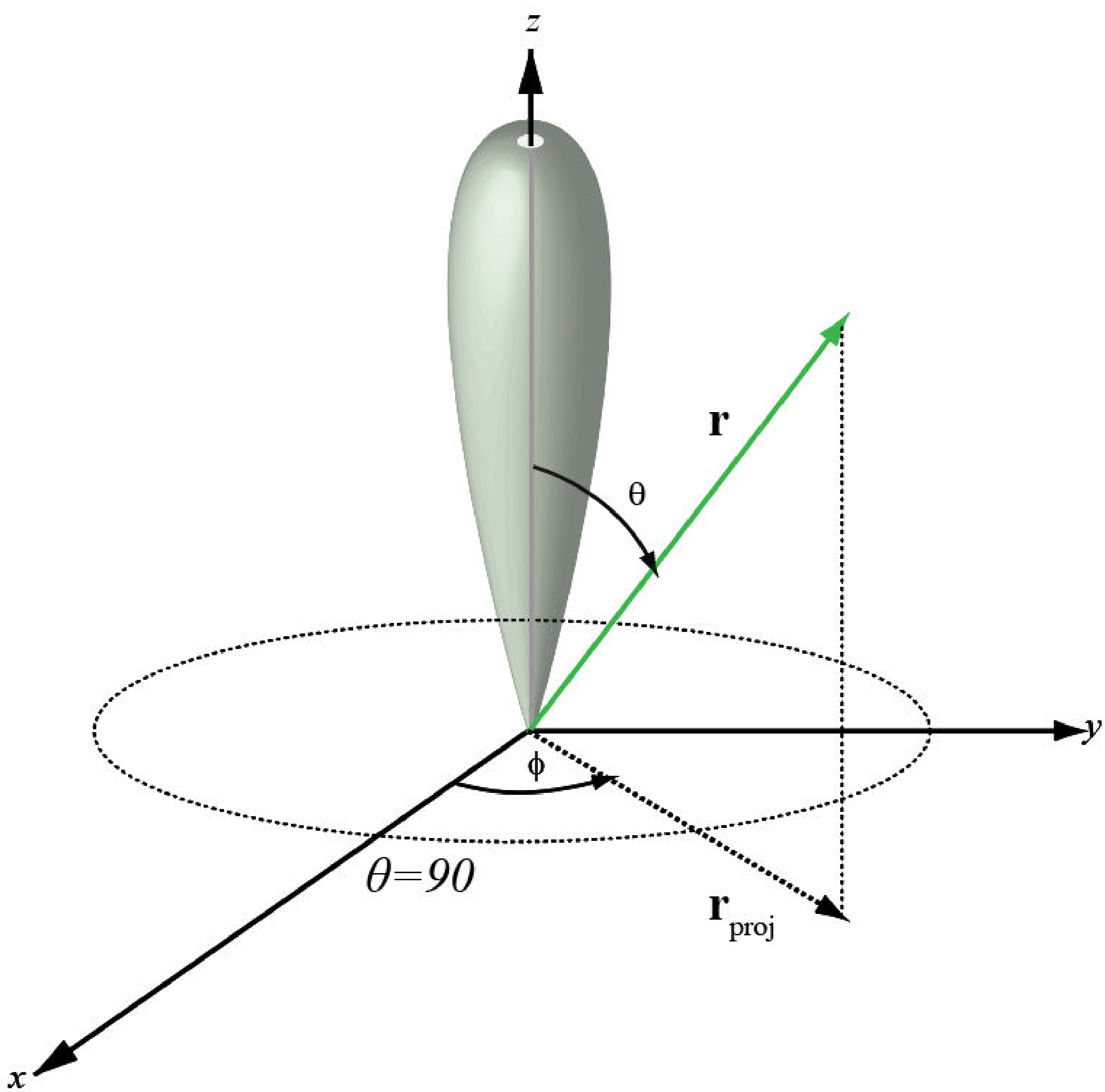
$\cos\theta = \cos el \cos az$

$\tan\phi = \tan el / \sin az$

Alternative Definition of Phi and Theta

The phi angle (φ) is the angle from the positive *x*-axis to the vector's orthogonal projection onto the *xy* plane. The angle is positive toward the positive *y*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *z*-axis to the vector itself. The angle is positive toward the *xy* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line.



$$\phi = az$$
$$\theta = 90 - el$$
$$az = \phi$$
$$el = 90 - \theta$$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phased.CustomAntennaElement` | `phitheta2azel` | `azel2phitheta` | `azel2phithetapat`

Topics

Antenna Array Analysis with Custom Radiation Pattern
"Spherical Coordinates"

phitheta2uv

Convert phi/theta angles to u/v coordinates

Syntax

```
UV = phitheta2uv(PhiTheta)
```

Description

`UV = phitheta2uv(PhiTheta)` converts the phi/theta angle on page 2-276 pairs to their corresponding *u*/*v* space on page 2-277 coordinates.

Examples

Conversion of Phi-Theta Pair

Find the corresponding *u*-*v* representation for φ = 30° and φ = 0°.

```
uv = phitheta2uv([30; 0])
```

```
uv = 2×1

     0
     0
```

Input Arguments

`PhiTheta` — Phi/theta angle pairs

two-row matrix

Phi and theta angles, specified as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [phi; theta].

Data Types: `double`

Output Arguments

UV — Angle in u/v space

two-row matrix

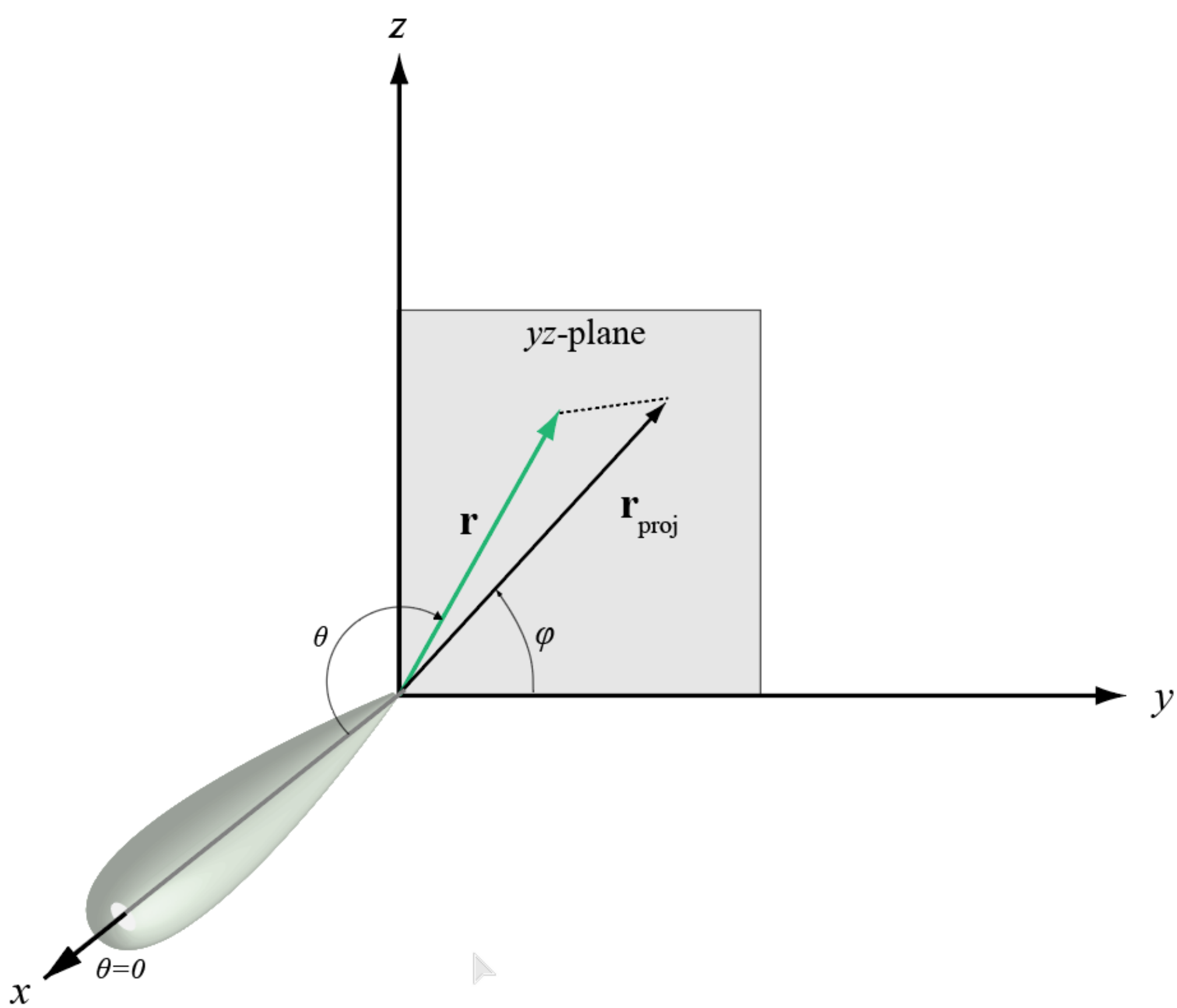
Angle in *u*/*v* space, returned as a two-row matrix. Each column of the matrix represents an angle in the form [*u*; *v*]. The matrix dimensions of `UV` are the same as those of `PhiTheta`.

More About

Phi Angle, Theta Angle

The phi angle (φ) is the angle from the positive *y*-axis to the vector's orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin el = \sin\phi\sin\theta$$
$$\tan az = \cos\phi\tan\theta$$
$$\cos\theta = \cos el\cos az$$
$$\tan\phi = \tan el/\sin az$$

U/V Space

The *u/v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles on page 2-276.

The relations are

$$u = \sin\theta\cos\phi$$
$$v = \sin\theta\sin\phi$$

In these expressions, φ and θ are the phi and theta angles, respectively.

To convert azimuth and elevation to *u* and *v* use the transformation

$$u = \cos el\sin az$$
$$v = \sin el$$

which is valid only in the range *abs(az)≤=90*.

The values of *u* and *v* satisfy the inequalities

$$-1 \leq u \leq 1$$
$$-1 \leq v \leq 1$$
$$u^2 + v^2 \leq 1$$

Conversely, the phi and theta angles can be written in terms of *u* and *v* using

$$\tan\phi = v/u$$
$$\sin\theta = \sqrt{u^2 + v^2}$$

The azimuth and elevation angles can also be written in terms of *u* and *v*:

$$\sin el = v$$
$$\tan az = \frac{u}{\sqrt{1 - u^2 - v^2}}$$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`uv2phitheta`

Topics
"Spherical Coordinates"

phitheta2uvpat

Convert radiation pattern from phi/theta form to u/v form

Syntax

```
pat_uv = phitheta2uvpat(pat_phitheta,phi,theta)
pat_uv = phitheta2uvpat(pat_phitheta,phi,theta,u,v)
[pat_uv,u_pat,v_pat] = phitheta2uvpat( ___ )
```

Description

`pat_uv = phitheta2uvpat(pat_phitheta,phi,theta)` expresses the antenna radiation pattern `pat_phitheta` in u/v space on page 2-284 coordinates instead of φ/θ angle on page 2-283 coordinates. `pat_phitheta` samples the pattern at φ angles in `phi` and θ angles in `theta`. The `pat_uv` matrix uses a default grid that covers *u* values from –1 to 1 and *v* values from –1 to 1. In this grid, `pat_uv` is uniformly sampled with a step size of 0.01 for *u* and *v*. The function interpolates to estimate the response of the antenna at a given direction. Values in `pat_uv` are NaN for *u* and *v* values outside the unit circle because *u* and *v* are undefined outside the unit circle.

`pat_uv = phitheta2uvpat(pat_phitheta,phi,theta,u,v)` uses vectors `u` and `v` to specify the grid at which to sample `pat_uv`. To avoid interpolation errors, `u` should cover the range [–1, 1] and `v` should cover the range [–1, 1].

`[pat_uv,u_pat,v_pat] = phitheta2uvpat( ___ )` returns vectors containing the *u* and *v* coordinates at which `pat_uv` samples the pattern, using any of the input arguments in the previous syntaxes.

Examples

Conversion of Radiation Pattern

Convert a radiation pattern to *u*-*v* form, with the *u* and *v* coordinates spaced by 0.01.

Define the pattern in terms of φ and θ.

```
phi = 0:360;
theta = 0:90;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Convert the pattern to *u*-*v* form.

```
pat_uv = phitheta2uvpat(pat_phitheta,phi,theta);
```

Convert and Plot Radiation Pattern

Convert a radiation pattern to $u-v$ coordinates, with the *u* and *v* coordinates spaced by 0.01.

Define the pattern in terms of ϕ and θ.

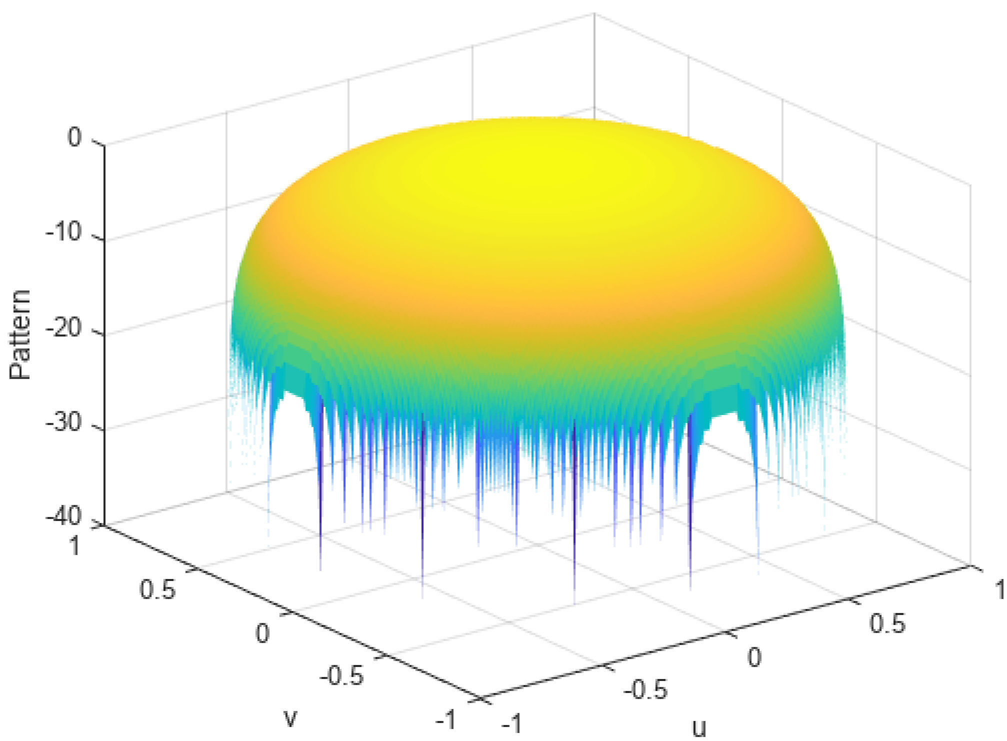
```
phi = 0:360;
theta = 0:90;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Convert the pattern to $u-v$ coordinates. Store the u and v coordinates for use in plotting.

```
[pat_uv,u,v] = phitheta2uvpat(pat_phitheta,phi,theta);
```

Plot the result.

```
H = surf(u,v,pat_uv);
H.LineStyle = 'none';
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



Convert Radiation Pattern for Specific U/V Values

Convert a radiation pattern to $u-v$ coordinates, with the u and v coordinates spaced by 0.05.

Define the pattern in terms of ϕ and θ.

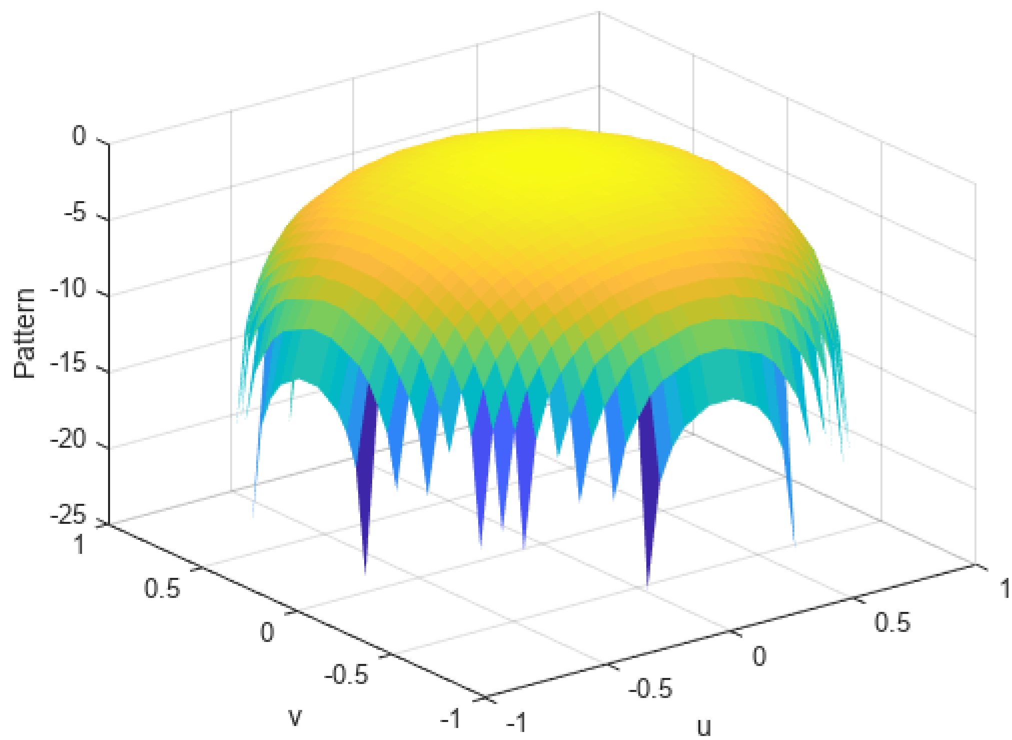
```
phi = 0:360;
theta = 0:90;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Define the set of *u* and *v* coordinates at which to sample the pattern. Then, convert the pattern.

```
u = -1:0.05:1;
v = -1:0.05:1;
pat_uv = phitheta2uvpat(pat_phitheta,phi,theta,u,v);
```

Plot the result.

```
H = surf(u,v,pat_uv);
H.LineStyle = 'none';
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



Input Arguments

pat_phitheta — Antenna radiation pattern in phi/theta form
Q-by-P matrix

Antenna radiation pattern in phi/theta form, specified as a Q-by-P matrix. `pat_phitheta` samples the 3-D magnitude pattern in decibels, in terms of φ and θ angles. P is the length of the `phi` vector, and Q is the length of the `theta` vector.

Data Types: double

phi — Phi angles
vector of length P

Phi angles at which pat_phitheta samples the pattern, specified as a vector of length P. Each φ angle is in degrees, between 0 and 180.

Data Types: double

theta — Theta angles
vector of length Q

Theta angles at which pat_phitheta samples the pattern, specified as a vector of length Q. Each θ angle is in degrees, between 0 and 90. Such angles are in the hemisphere for which *u* and *v* are defined.

Data Types: double

u — *u* coordinates
[-1:0.01:1] (default) | vector of length L

u coordinates at which pat_uv samples the pattern, specified as a vector of length L. Each *u* coordinate is between –1 and 1.

Data Types: double

v — *v* coordinates
[-1:0.01:1] (default) | vector of length M

v coordinates at which pat_uv samples the pattern, specified as a vector of length M. Each *v* coordinate is between –1 and 1.

Data Types: double

Output Arguments

pat_uv — Antenna radiation pattern in *u*/*v* form
M-by-L matrix

Antenna radiation pattern in *u*/*v* form, returned as an M-by-L matrix. pat_uv samples the 3-D magnitude pattern in decibels, in terms of *u* and *v* coordinates. L is the length of the u vector, and M is the length of the v vector. Values in pat_uv are NaN for *u* and *v* values outside the unit circle because *u* and *v* are undefined outside the unit circle.

u_pat — *u* coordinates
vector of length L

u coordinates at which pat_uv samples the pattern, returned as a vector of length L.

v_pat — *v* coordinates
vector of length M

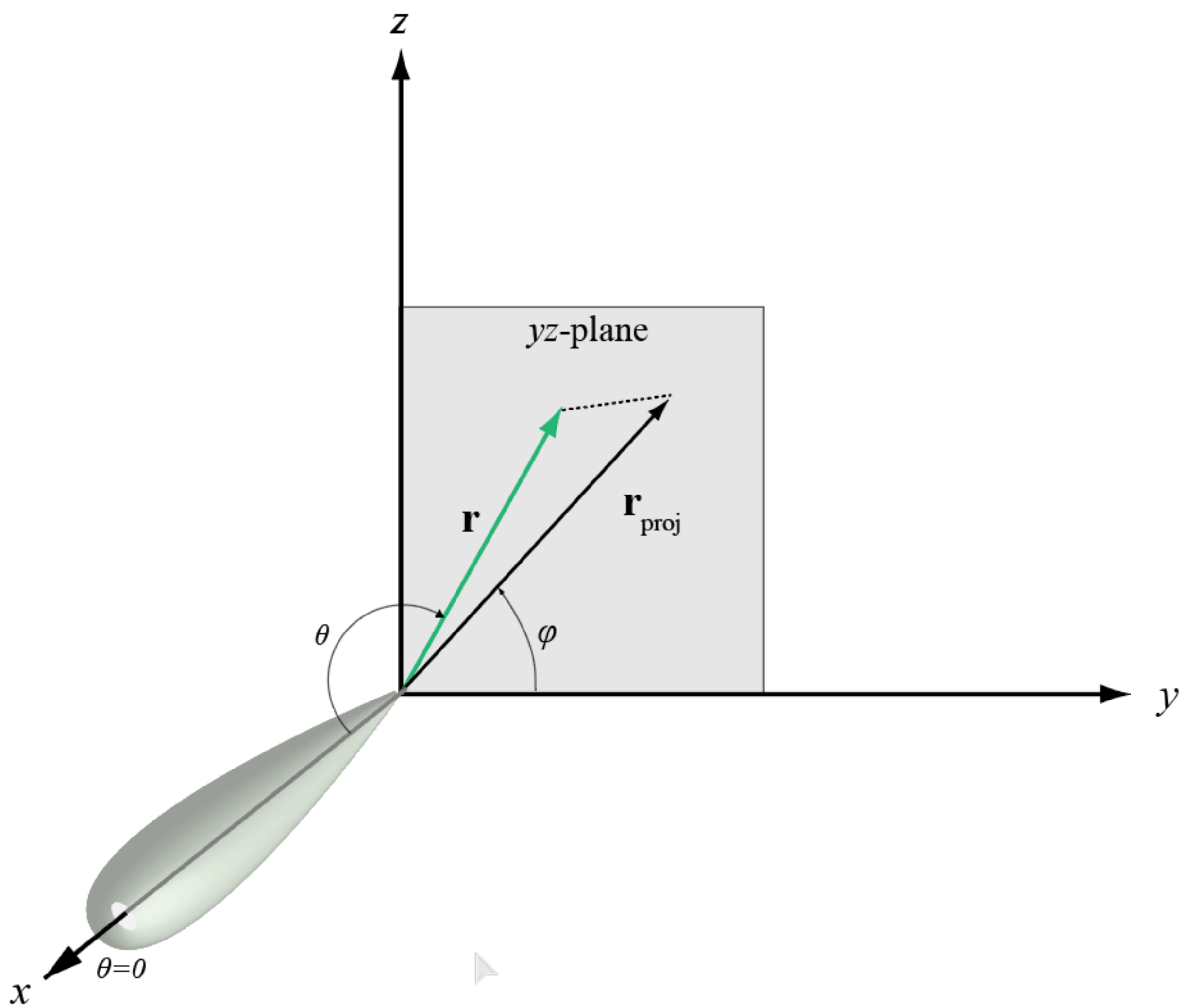
v coordinates at which pat_uv samples the pattern, returned as a vector of length M.

More About

Phi Angle, Theta Angle

The phi angle (φ) is the angle from the positive *y*-axis to the vector's orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi \sin\theta$
$\tan az = \cos\phi \tan\theta$
$\cos\theta = \cos el \cos az$
$\tan\phi = \tan el/\sin az$

U/V Space

The *u* and *v* coordinates are the direction cosines of a vector with respect to the *y*-axis and *z*-axis, respectively.

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles on page 2-283, as follows:

$u = \sin\theta \cos\phi$
$v = \sin\theta \sin\phi$

In these expressions, φ and θ are the phi and theta angles, respectively.

To convert azimuth and elevation to *u* and *v* use the transformation

$u = \cos el \sin az$
$v = \sin el$

which is valid only in the range *abs(az)≤=90*.

The values of *u* and *v* satisfy the inequalities

$-1 \leq u \leq 1$
$-1 \leq v \leq 1$
$u^2 + v^2 \leq 1$

Conversely, the phi and theta angles can be written in terms of *u* and *v* using

$\tan\phi = v/u$
$\sin\theta = \sqrt{u^2 + v^2}$

The azimuth and elevation angles can also be written in terms of *u* and *v*:

$\sin el = v$

$$\tan az = \frac{u}{\sqrt{1 - u^2 - v^2}}$$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phased.CustomAntennaElement` | `phitheta2uv` | `uv2phitheta` | `uv2phithetapat`

Topics

"Spherical Coordinates"

physconst

Physical constants

Syntax

```
const = physconst(name)
```

Description

`const = physconst(name)` returns the value of the physical constant `const` specified by the `name` argument.

Examples

Convert Frequency to Wavelength

Determine the wavelength of a 1 GHz electromagnetic wave.

```
freq = 1e9;
lambda = physconst('LightSpeed')/freq
```

```
lambda = 0.2998
```

Thermal Noise Power

Determine the thermal noise power per unit bandwidth in the in-phase (I) and quadrature (Q) channels of a receiver. Specify a receiver temperature of 290 K.

```
T = 290;
k = physconst('Boltzmann');
```

Compute the noise power per unit bandwidth, split evenly between the I and Q channels. Units are in dB.

```
Noise_power = 10*log10(k*T/2)
```

```
Noise_power = -206.9855
```

Input Arguments

name — Name of physical constant
'LightSpeed' | 'Boltzmann' | 'EarthRadius'

Name of physical constant, specified as `'LightSpeed'`, `'Boltzmann'`, or `'EarthRadius'`. See “Physical Constants” on page 2-287 for a list of values for physical constants used in Phased Array System Toolbox.

Example: `'LightSpeed'`

Output Arguments

`const` — Value of physical constant
real-valued scalar

Value of physical constant, returned as a real-valued scalar. All values are in SI units.

More About

Physical Constants

This table lists the supported constants and their values in SI units.

Constant	Description	Value
`'LightSpeed'`	Speed of light in vacuum	299,792,458 m/s. Most commonly denoted by *c*.
`'Boltzmann'`	Boltzmann constant relating kinetic energy to temperature	$1.3806504 \times 10^{-23}$ J/K. 2006 NIST value, most commonly denoted by *k*.
`'EarthRadius'`	Mean radius of the Earth	6,371,000 m

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

The function does not support variable-size inputs.

See Also

External Websites

https://physics.nist.gov/cuu/Constants/index.html

pilotcalib

Array calibration using pilot sources

Syntax

```
estpos = pilotcalib(nompos,x,pilotang)
[estpos,esttaper] = pilotcalib(nompos,x,pilotang)
[estpos,esttaper] = pilotcalib(nompos,x,pilotang,nomtaper)
[estpos,esttaper] = pilotcalib(nompos,x,pilotang,nomtaper,uncerts)
```

Description

`estpos = pilotcalib(nompos,x,pilotang)` returns the estimated element positions, `estpos`, of a sensor array. The argument `nompos` represents the relative nominal positions of the sensor array before calibration. The nominal position is relative to the first element of the array. The argument `x` represents the signals received by the array coming from the pilot sources. The argument `pilotang` contains the known directions of each of the pilot sources. Three or more pilot sources are required in this case.

`[estpos,esttaper] = pilotcalib(nompos,x,pilotang)` also returns the estimated array taper, `esttaper`. Each element of `esttaper` contains the estimated taper value of the corresponding array element. In this case, the prior nominal taper is one for each element. Four or more pilot sources are required in this case.

`[estpos,esttaper] = pilotcalib(nompos,x,pilotang,nomtaper)` specifies `nomtaper` as the nominal taper of the array. Four or more pilot sources are required in this case.

`[estpos,esttaper] = pilotcalib(nompos,x,pilotang,nomtaper,uncerts)` specifies `uncerts` as the configuration settings to use for calibrating array uncertainties. Configuration settings determine which parameters to estimate.

Examples

Estimate ULA Element Positions Using Pilot Calibration

Construct a 7-element ULA array of isotropic antenna elements spaced one-half wavelength apart. Assume the array is geometrically perturbed in three dimensions. Perform pilot calibration on the array using 4 pilot sources at azimuth and elevation angles of (-60,0), (10,-40), (40,0), and (120,45) degrees. For the calibration process, pilot signals have an SNR of 30 dB. Each pilot signal contains 10,000 samples. Assume the signals have a frequency of 600 MHz.

Set up the ULA with nominal parameters

```
fc = 600e6;
c = physconst('LightSpeed');
lam = c/fc;
d = 0.5*lam;
sIso = phased.IsotropicAntennaElement('FrequencyRange',[100,900]*1e6);
Nelem = 7;
```

```
NominalTaper = ones(1,Nelem);
sULA = phased.ULA('Element',sIso,'NumElements',Nelem,'ElementSpacing',d,...
    'Taper',NominalTaper);
```

Create the pilot signals

Randomly perturb the element positions with a Gaussian distribution having 0.1 wavelength standard deviation. Do not perturb the position of the first element or the tapers.

```
posstd = 0.1;
rng default
NominalElementPositions = getElementPosition(sULA)/lam;
ReferenceElement = NominalElementPositions(:,1);
PositionPert = [zeros(3,1),posstd*randn(3,Nelem-1)];
ActualElementPositions = NominalElementPositions + PositionPert;
ActualTaper = NominalTaper;
```

Generate the signals using the actual positions and tapers.

```
Nsamp  = 10000;
ncov = 0.001;
PilotAng = [-60,10,40,120; 0,-40,0,45];
Npilot = size(PilotAng,2);
for n = 1:Npilot
    X(:,:,n) = sensorsig(ActualElementPositions,...
        Nsamp,PilotAng(:,n),ncov,'Taper',ActualTaper.');
end
```

Perform the pilot calibration

```
estpos = pilotcalib(NominalElementPositions - ReferenceElement*ones(1,Nelem),...
    X,PilotAng);
```

Add back the position of the reference sensor.

```
estpos = estpos + NominalElementPositions(:,1)*ones(1,Nelem);
```

Examine the root mean squared (RMS) error of the calibrated parameters

Compute the RMS value of the initial position error.

```
numpos = 3*Nelem;
initposRMSE = sqrt(sum(PositionPert(:).^2)/numpos);
```

Compute the RMS value of the calibrated position error.

```
solvposErr = ActualElementPositions - estpos;
solvposRMSE = sqrt(sum(solvposErr(:).^2)/(numpos));
```

Compare the calibrated RMS position error to the initial position RMS error. The calibration reduces the RMS position error.

```
disp(solvposRMSE/initposRMSE)
```

```
   2.3493e-04
```

Estimate ULA Element Position and Taper Errors Using Pilot Calibration

Construct a 7-element ULA array of isotropic antenna elements spaced one-half wavelength apart. Assume the array is geometrically perturbed in three dimensions. Perform pilot calibration on the array using 4 pilot sources at azimuth and elevation angles of (-60,0), (10,80), (40,-40), and (-80,0) degrees. For the calibration process, pilot signals have an SNR of 30 dB. Each pilot signal contains 10,000 samples. Assume the signals have a frequency of 600 MHz.

Set up the ULA with nominal parameters

```
fc = 600e6;
c = physconst('LightSpeed');
lam = c/fc;
d = 0.5*lam;
sIso = phased.IsotropicAntennaElement('FrequencyRange',[100,900]*1e6);
Nelem = 7;
NominalTaper = ones(1,Nelem);
sULA = phased.ULA('Element',sIso,'NumElements',Nelem,'ElementSpacing',d,...
    'Taper',NominalTaper);
```

Create the pilot signals

Randomly perturb the element positions using a Gaussian distribution that has a standard deviation of 0.1 wavelength. Do not perturb the position of the first element.

```
posstd = 0.1;
rng default
NominalElementPositions = getElementPosition(sULA)/lam;
ReferenceElement = NominalElementPositions(:,1);
PositionPert = [zeros(3,1),posstd*randn(3,Nelem-1)];
ActualElementPositions = NominalElementPositions + PositionPert;
```

Perturb the taper in magnitude and phase. Do not perturb the first taper.

```
tapermagstd = 0.15;
taperphasestd = 0.15;
tapermagpert = tapermagstd*[0; randn(Nelem-1,1)];
ActualTaper = NominalTaper' + tapermagpert;
taperphasepert = taperphasestd*[0;randn(Nelem-1,1)];
ActualTaper = ActualTaper.*exp(1i*taperphasepert);
```

Generate the signals using the perturbed positions, tapers and four pilot sources.

```
Nsamp  = 10000;
ncov = 0.001;
PilotAng = [-60,10,40,-80; 10,80,-40,0];
Npilot = size(PilotAng,2);
for n = 1:Npilot
    X(:,:,n) = sensorsig(ActualElementPositions,Nsamp,...,
        PilotAng(:,n),ncov,'Taper',ActualTaper);
end
```

Perform the pilot calibration

```
[estpos,esttaper] = pilotcalib(...
    NominalElementPositions - ReferenceElement*ones(1,Nelem),...
    X,PilotAng);
```

Add back the position of the reference sensor.

```
estpos = estpos + NominalElementPositions(:,1)*ones(1,Nelem);
```

Examine the root mean square (RMS) error of the calibrated parameters

Compute the RMS values of the initial taper perturbations.

```
tapermagpertRMSE = sqrt(tapermagpert'*tapermagpert/Nelem);
taperphasepertRMSE = sqrt(taperphasepert'*taperphasepert/Nelem);
```

Compute the RMS value of the calibrated taper magnitude error.

```
diff = abs(ActualTaper) - abs(esttaper);
diff2 = diff'*diff;
tapermagsolvRMSE = sqrt(diff2/Nelem);
```

Compare the calibrated RMS magnitude error to the initial RMS magnitude error. The calibration reduces the RMS magnitude error.

```
disp(tapermagsolvRMSE/tapermagpertRMSE)
```

```
   6.7715e-04
```

Compute the RMS value of the calibrated taper phase error.

```
diff = unwrap(angle(ActualTaper) - angle(esttaper));
diff2 = diff'*diff;
tapersolvphaseRMSE = sqrt(diff2/Nelem);
```

Compare the calibrated RMS phase error to the initial RMS phase error. The calibration reduces the RMS phase error.

```
disp(tapersolvphaseRMSE/taperphasepertRMSE)
```

```
    0.0021
```

```
% Compute the RMS value of the initial position error.
numpos = 3*Nelem;
initposRMSE = sqrt(sum(PositionPert(:).^2)/numpos);
```

Compute the RMS value of the calibrated position error.

```
solvposErr = ActualElementPositions - estpos;
solvposRMSE = sqrt(sum(solvposErr(:).^2)/(numpos));
```

Compare the calibrated RMS position error to the initial position RMS error. The calibration reduces the RMS position error.

```
disp(solvposRMSE/initposRMSE)
```

```
   3.6308e-04
```

Estimate URA Element Position Errors Using Pilot Calibration

Construct a 9-element URA of isotropic antenna elements spaced one-half wavelength apart. Assume the array has been geometrically perturbed in all directions except for the first element. Perform pilot calibration on the array using 5 pilot sources at azimuth and elevation angles of (-60,0), (10,-40), (40,0), (120,45), and (170,50) degrees. For the calibration process, pilot signals have an SNR of 30 dB. Each pilot signal contains 10,000 samples. Assume the signals have a frequency of 600 MHz.

Create the array

For convenience, use a `phased.URA` System object™ to set the nominal position and taper values.

```
fc = 300e6;
c = physconst('LightSpeed');
lam = c/fc;
d = 0.5*lam;
sIso = phased.IsotropicAntennaElement('FrequencyRange',[100,900]*1e6);
sURA = phased.URA('Element',sIso,'Size',[3,3],...
    'ElementSpacing',d,'Taper',ones(3,3));
Nelem = getNumElements(sURA);
taper = getTaper(sURA);
```

Create the pilot signals

Randomly perturb the element positions using a Gaussian distribution that has a standard deviation of 0.1 wavelength. Do not perturb the position of the first element.

```
posstd = 0.1;
rng default
NominalElementPositions = getElementPosition(sURA)/lam;
ReferenceElement = NominalElementPositions(:,1);
PositionPert = [zeros(3,1),posstd*randn(3,Nelem-1)];
ActualElementPositions = NominalElementPositions + PositionPert;
```

Perturb the taper in magnitude and phase. Do not perturb the first taper.

```
NominalTaper = getTaper(sURA);
tapermagstd = 0.1;
taperphasestd = 0.1;
tapermagpert = tapermagstd*[0; randn(Nelem-1,1)];
ActualTaper = NominalTaper + tapermagpert;
taperphasepert = taperphasestd*[0;randn(Nelem-1,1)];
ActualTaper = ActualTaper.*exp(1i*taperphasepert);
```

Generate the pilot signals using the perturbed positions and tapers.

```
Nsamp  = 10000;
ncov = 0.001;
PilotAng = [-60,10,40,120,170; 0,-40,0,45,50];
Npilot = size(PilotAng,2);
for n = 1:Npilot
    X(:,:,n) = sensorsig(ActualElementPositions,Nsamp,...
        PilotAng(:,n),ncov,'Taper',ActualTaper);
end
```

Perform the pilot calibration

```
[estpos,esttaper] = pilotcalib(NominalElementPositions - ReferenceElement*ones(1,Nelem),...
    X,PilotAng,NominalTaper);
```

Add back the position of the reference sensor.

```
estpos = estpos + NominalElementPositions(:,1)*ones(1,Nelem);
```

Examine the root mean square (RMS) error of the calibrated parameters

Compute the RMS values of the initial taper perturbations to compare with the RMS values of the calibrated parameters.

```
tapermagpertRMSE = sqrt(tapermagpert'*tapermagpert/Nelem);
taperphasepertRMSE = sqrt(taperphasepert'*taperphasepert/Nelem);
```

Compute the RMS value of the calibrated taper magnitude error.

```
diff = abs(ActualTaper) - abs(esttaper);
diff2 = diff'*diff;
tapermagsolvRMSE = sqrt(diff2/Nelem);
```

Compare the calibrated RMS magnitude error to the initial RMS error. The calibration reduces the RMS magnitude error.

```
disp(tapermagsolvRMSE/tapermagpertRMSE)
```

```
    0.0014
```

Compute the RMS value of the calibrated taper phase error.

```
diff = unwrap(angle(ActualTaper) - angle(esttaper));
diff2 = diff'*diff;
tapersolvphaseRMSE = sqrt(diff2/Nelem);
```

Compare the calibrated RMS phase error to the initial RMS error. The calibration reduces the RMS phase error.

```
disp(tapersolvphaseRMSE/taperphasepertRMSE)
```

```
    0.0015
```

Compute the RMS value of the initial position error.

```
numpos = 3*Nelem;
initposRMSE = sqrt(sum(PositionPert(:).^2)/numpos);
```

Compute the RMS value of the calibrated position error.

```
solvposErr = ActualElementPositions - estpos;
solvposRMSE = sqrt(sum(solvposErr(:).^2)/(numpos));
```

Compare the calibrated RMS position error to initial position RMS error. The calibration reduces the RMS position error.

```
disp(solvposRMSE/initposRMSE)
```

```
   7.1582e-04
```

Estimate Selected ULA Parameters Using Pilot Calibration

Construct a 6-element ULA of isotropic antenna elements that are spaced one-half wavelength apart. Assume the array has been geometrically perturbed in the *x*-*y* plane and contains an unknown taper error. Perform pilot calibration on the array using four pilot sources at azimuth and elevation angles of (-60,0), (10,-40), (40,0), and (120,45) degrees. For the calibration process, pilot signals have an

SNR of 30 dB. Each pilot signal contains 10,000 samples. Assume the signals have a frequency of 600 MHz.

Set up the ULA with nominal parameters

```
fc = 600e6;
c = physconst('LightSpeed');
lam = c/fc;
d = 0.5*lam;
sIso = phased.IsotropicAntennaElement('FrequencyRange',[100,900]*1e6);
Nelem = 6;
NominalTaper = ones(1,Nelem);
sULA = phased.ULA('Element',sIso,'NumElements',Nelem,'ElementSpacing',d,...
    'Taper',NominalTaper);
```

Create the pilot signals

Randomly perturb the element positions using a Gaussian distribution that has a standard deviation of 0.13 wavelength. Do not perturb the position of the first element.

```
posstd = 0.13;
rng default
NominalElementPositions = getElementPosition(sULA)/lam;
ReferenceElement = NominalElementPositions(:,1);
PositionPert = [zeros(3,1),posstd*randn(3,Nelem-1)];
ActualElementPositions = NominalElementPositions + PositionPert;
```

Perturb the taper in magnitude and phase. Do not perturb the first taper.

```
tapermagstd = 0.15;
taperphasestd = 0.15;
tapermagpert = tapermagstd*[0; randn(Nelem-1,1)];
ActualTaper = NominalTaper' + tapermagpert;
taperphasepert = taperphasestd*[0;randn(Nelem-1,1)];
ActualTaper = ActualTaper.*exp(1i*taperphasepert);
```

Generate the signals using the perturbed positions and tapers.

```
Nsamp  = 10000;
ncov = 0.001;
PilotAng = [-60,10,40,120; 0,-40,0,45];
Npilot = size(PilotAng,2);
for n = 1:Npilot
    X(:,:,n) = sensorsig(ActualElementPositions,Nsamp,...
        PilotAng(:,n),ncov,'Taper',ActualTaper);
end
```

Perform the pilot calibration

Turn off estimation of taper weights.

```
[estpos,esttaper] = pilotcalib(NominalElementPositions - ReferenceElement*ones(1,Nelem),...
    X,PilotAng,NominalTaper.',[1,1,1,0]');
```

Add back the position of the reference sensor.

```
estpos = estpos + NominalElementPositions(:,1)*ones(1,Nelem);
```

Examine the root mean square (RMS) error of the calibrated parameters

Compute the RMS values of the initial taper perturbations to compare with the RMS values of the calibrated parameters.

```
tapermagpertRMSE = sqrt(tapermagpert'*tapermagpert/Nelem);
taperphasepertRMSE = sqrt(taperphasepert'*taperphasepert/Nelem);
```

Compute the RMS value of the calibrated taper magnitude error.

```
diff = abs(ActualTaper) - abs(esttaper);
diff2 = diff'*diff;
tapermagsolvRMSE = sqrt(diff2/Nelem);
```

Compare the calibrated RMS magnitude error to the initial RMS error. The calibration reduces the RMS magnitude error.

```
disp(tapermagsolvRMSE/tapermagpertRMSE)
```

```
    1.0000
```

Compute the RMS value of the calibrated taper phase error.

```
diff = unwrap(angle(ActualTaper) - angle(esttaper));
diff2 = diff'*diff;
tapersolvphaseRMSE = sqrt(diff2/Nelem);
```

Compare the calibrated RMS phase error to the initial RMS error. The calibration reduces the RMS phase error.

```
disp(tapersolvphaseRMSE/taperphasepertRMSE)
```

```
     1
```

Compute the RMS value of the initial position error.

```
numpos = 3*Nelem;
initposRMSE = sqrt(sum(PositionPert(:).^2)/numpos);
```

Compute the RMS value of the calibrated position error.

```
solvposErr = ActualElementPositions - estpos;
solvposRMSE = sqrt(sum(solvposErr(:).^2)/(numpos));
```

Compare the calibrated RMS position error to initial position RMS error. The calibration reduces the RMS position error.

```
disp(solvposRMSE/initposRMSE)
```

```
    0.1502
```

Input Arguments

nompos — Nominal relative element positions
real-valued 3-by-*N* matrix

Nominal relative element positions, specified as a real-valued 3-by-*N* matrix. The dimension *N* is the number of elements in the sensor array. Elements positions are relative to the first element of the array and are specified in units of signal wavelength. Each column of `nompos` represents the `[x;y;z]` coordinates of the corresponding element. The nominal position of all sensors must be within one-half of a wavelength of their actual positions for successful calibration.

Data Types: `double`

`x` — Pilot signals
complex-valued *L*-by-*N*-by-*M* matrix

Pilot signals, specified as a complex-valued *L*-by-*N*-by-*M* matrix. The argument `x` represents the signals received by the array when pilot sources are transmitting. The dimension *L* is the number of snapshots of each pilot source signal. The dimension *N* is the number of array elements. The dimension *M* is the number of pilot sources.

Data Types: `double`
Complex Number Support: Yes

`pilotang` — Pilot angles
real-valued 2-by-*M* matrix

Pilot angles, specified as a real-valued 2-by-*M* matrix. The dimension *M* is the number of pilot sources. Each column contains the direction of the pilot source in the form `[azimuth; elevation]`. Angle units are in degrees. The azimuth angle must lie between -180° and 180° and the elevation angle must lie between -90° and 90°. The azimuth angle is measured from the *x*-axis to the projection of the source direction into the *xy* plane, positive toward the *y*-axis. The elevation angle is defined as the angle from the *xy* plane to the source direction, positive toward the *z*-axis. Calibration source directions must span sufficiently diverse azimuth and elevation angles.

Data Types: `double`

`nomtaper` — Nominal taper
1 (default) | complex-valued *N*-by-1 column vector

Nominal taper of array elements, specified as a complex-valued *N*-by-1 column vector. The dimension *N* is the number of array elements. Each component represents the nominal taper of the corresponding element.

Data Types: `double`
Complex Number Support: Yes

`uncerts` — Uncertainty estimation configuration
`[1,1,1,1]` (default) | 1-by-4 vector of ones and zeros

Uncertainty estimation configuration, specified as a 1-by-4 vector consisting of `0`'s and `1`'s. The vector `uncerts` determines which uncertainties to estimated. The vector takes the form of `[xflag; yflag; zflag; taperflag]`. Set `xflag`, `yflag`, or `zflag` to `1` to estimate uncertainties in the *x*, *y*, or *z* axes. Set `taperflag` to `1` to estimate uncertainties in the taper. The number of pilot sources must greater than or equal to the number of `1`'s in the vector.

For example, set `uncerts` to `[0;1;1;1]` to estimate uncertainties in the *y* and *z* element position components and the taper simultaneously.

Data Types: `double`

Output Arguments

`estpos` — Estimated positions
real-valued 3-by-*N* matrix

Estimated element positions, returned as a real-valued 3-by-*N* matrix. Units are in signal wavelength. The dimension *N* is the number of array elements. Each column of `estpos` represents the `[x;y;z]` coordinates of the corresponding element.

`esttaper` — Estimated taper
complex-valued *N*-by-1 column vector

Estimated taper values, returned as a complex-valued *N*-by-1 column vector. The dimension *N* is the number of array elements. Each element of `esttaper` represents the taper of the corresponding sensor element.

Algorithms

This algorithm requires that the pilot sources be independent narrowband plane-wave signals incoming from the far field region of the array. In addition, signals must not exhibit multipath propagation effects or coherence. All elements in the sensor array are assumed to be isotropic.

The algorithm calibrates relative positions of the array sensors with respect to the first sensor. To use the algorithm, first subtract the position of the first element from each element, then pass the relative array into the function as the nominal position argument to produced the calibrated relative positions. Finally, add back the first element position to all the relative positions to create the fully calibrated array.

Version History

Introduced in R2015a

References

[1] N. Fistas and A. Manikas, "A New General Global Array Calibration Method", *IEEE Proceedings of ICASSP*, Vol. IV, pp. 73-76, April 1994.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

pol2circpol

Convert linear component representation of field to circular component representation

Syntax

```
cfv = pol2circpol(fv)
```

Description

`cfv = pol2circpol(fv)` converts the linear polarization components of the field or fields contained in `fv` to their equivalent circular polarization components in `cfv`. The expression of a field in terms of a two-row vector of linear polarization components is called the Jones vector formalism.

Examples

Convert Linear to Circular Polarization Components

Express a 45° linear polarized field in terms of right-circular and left-circular components.

```
fv = [2;2]
```

```
fv = 2×1

     2
     2
```

```
cfv = pol2circpol(fv)
```

```
cfv = 2×1 complex

   1.4142 - 1.4142i
   1.4142 + 1.4142i
```

Convert Linear Polarization Components to Circular Polarization Components

Specify two input fields `[1+1i;-1+1i]` and `[1;1]` in the same matrix. The first field is a linear representation of a left-circularly polarized field and the second is a linearly polarized field.

```
fv=[1+1i 1;-1+1i 1]
```

```
fv = 2×2 complex

   1.0000 + 1.0000i   1.0000 + 0.0000i
  -1.0000 + 1.0000i   1.0000 + 0.0000i
```

```
cfv = pol2circpol(fv)
```

```
cfv = 2×2 complex

   1.4142 + 1.4142i   0.7071 - 0.7071i
   0.0000 + 0.0000i   0.7071 + 0.7071i
```

Input Arguments

`fv` — Field vector in linear component representation

1-by-*N* complex-valued row vector or a 2-by-*N* complex-valued matrix

Field vector in its linear component representation specified as a 1-by-*N* complex row vector or a 2-by-*N* complex matrix. If `fv` is a matrix, each column in `fv` represents a field in the form of `[Eh;Ev]`, where `Eh` and `Ev` are the field's horizontal and vertical polarization components. If `fv` is a vector, each entry in `fv` is assumed to contain the polarization ratio, `Ev/Eh`. For a row vector, the value `Inf` designates the case when the ratio is computed for a field with `Eh = 0`.

Example: `[1;-i]`

Example: `2 + pi/3*i`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`cfv` — Field vector in circular component representation

1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix

Field vector in circular component representation returned as a 1-by-*N* complex-valued row vector or 2-by-*N*complex-valued matrix. `cfv` has the same dimensions as `fv`. If `fv` is a matrix, each column of `cfv` contains the circular polarization components, `[El;Er]`, of the field where `El` and `Er` are the left-circular and right-circular polarization components. If `fv` is a row vector, then `cfv` is also a row vector and each entry in `cfv` contains the circular polarization ratio, defined as `Er/El`.

Version History

Introduced in R2013a

References

[1] Mott, H., *Antennas for Radar and Communications*, John Wiley & Sons, 1992.

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`circpol2pol` | `polellip` | `polratio` | `stokes`

polellip

Parameters of ellipse traced out by tip of a polarized field vector

Syntax

```
tau = polellip(fv)
[tau,epsilon] = polellip(fv)
[tau,epsilon,ar] = polellip(fv)
[tau,epsilon,ar,rs] = polellip(fv)
```

```
polellip(fv)
```

Description

`tau = polellip(fv)` returns the tilt angle, in degrees, of the polarization ellipse of a field or set of fields specified in `fv`. `fv` contains the linear polarization components of a field in either one of two forms: (1) each column represents a field in the form of `[Eh;Ev]`, where `Eh` and `Ev` are the field's horizontal and vertical linear polarization components or (2) each column contains the polarization ratio, `Ev/Eh`. The expression of a field in terms of a two-row vector of linear polarization components is called the Jones vector formalism.

`[tau,epsilon] = polellip(fv)` returns, in addition, a row vector, `epsilon`, containing the ellipticity angle (in degrees) of the polarization ellipses. The ellipticity angle is the angle determined by the ratio of the length of the semi-minor axis to semi-major axis and lies in the range `[-45°,45°]`. This syntax can use any of the input arguments in the previous syntax.

`[tau,epsilon,ar] = polellip(fv)` returns, in addition, a row vector, `ar`, containing the axial ratios of the polarization ellipses. The axial ratio is defined as the ratio of the lengths of the semi-major axis of the ellipse to the semi-minor axis. This syntax can use any of the input arguments in the previous syntaxes.

`[tau,epsilon,ar,rs] = polellip(fv)` returns, in addition, a cell array of character vectors, `rs`, containing the rotation senses of the polarization ellipses. Each entry in the array is one of `'Linear'`, `'Left Circular'`, `'Right Circular'`, `'Left Elliptical'` or `'Right Elliptical'`. This syntax can use any of the input arguments in the previous syntaxes.

`polellip(fv)` plots the polarization ellipse of the field specified in `fv`. This syntax requires that `fv` have only one column. Unlike the returned arguments, the size of the drawn ellipse depends upon the magnitude of `fv`.

Examples

Tilt Angle for Linearly Polarized Field

Create an input field that is linearly polarized by setting both the horizontal and vertical components to have the same phase. Then, compute the tilt angle.

```
fv = [2;1];
tau = polellip(fv)
```

```
tau = 26.5651
```

For linear polarization, `tau` is computed using `tau = atan(fv(2)/fv(1))*180/pi`.

Tilt Angle and Ellipticity for Elliptically Polarized Field

Start with an elliptically polarized input field (the horizontal and vertical components differ in magnitude and in phase). Choose the phase difference to be 90°.

```
fv = [3*exp(-i*pi/2);1];
[tau,epsilon] = polellip(fv)
```

```
tau = 1.3156e-15

epsilon = 18.4349
```

The tilt vanishes because of the 90° phase difference between the horizontal and vertical components of the field.

Tilt Angle, Ellipticity and Axial Ratio for Elliptically Polarized Field

Start with an elliptically polarized input field (the horizontal and vertical components differ in magnitude and in phase). Choose the phase difference to be 60°.

```
fv = [2*exp(-i*pi/3);1];
[tau,epsilon,ar] = polellip(fv)
```

```
tau = 16.8450

epsilon = 21.9269

ar = -2.4842
```

The nonzero tilt occurs because of the 60° phase difference. The negative value of the axial ratio indicates left elliptical polarization.

Tilt Angle, Ellipticity, Axial Ratio and Rotation Sense for Elliptically Polarized Field

Start with an elliptically polarized input field (the horizontal and vertical components differ in magnitude and in phase). Choose the phase difference to be 60°.

```
fv = [2*exp(-i*pi/3);1];
[tau,epsilon,ar,rs] = polellip(fv)
```

```
tau = 16.8450

epsilon = 21.9269

ar = -2.4842
```

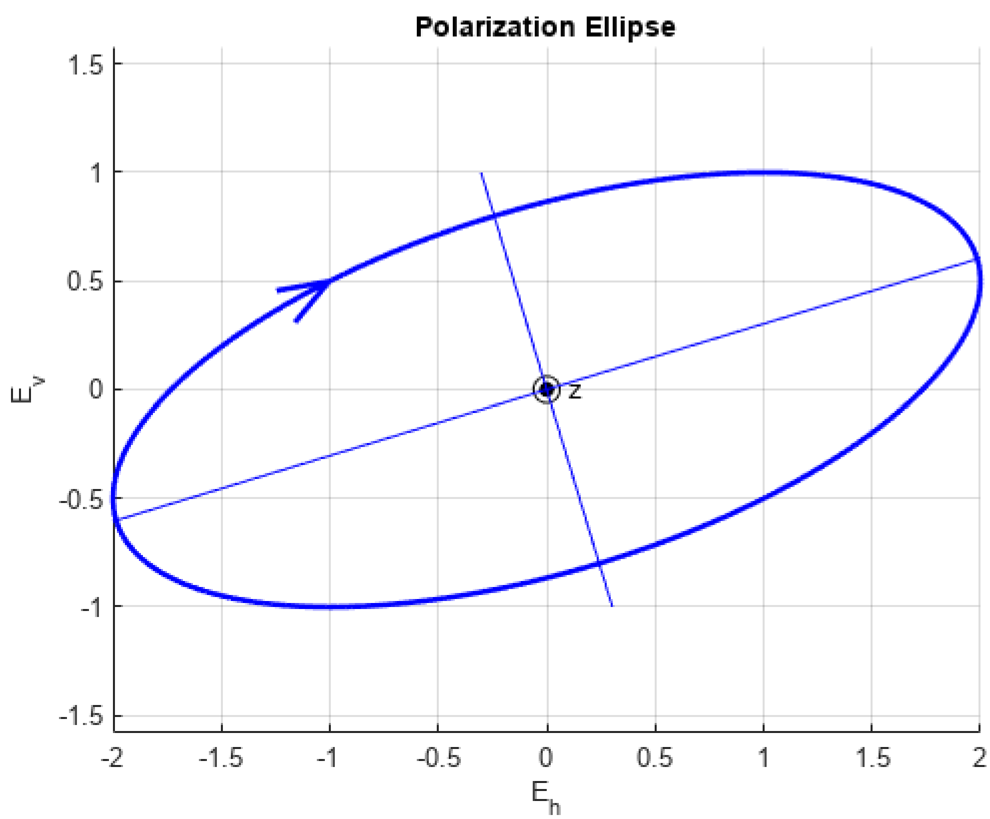
```
rs = 1x1 cell array
    {'Left Elliptical'}
```

A nonzero tilt occurs because of the 60° phase difference. The rotation sense is `'Left Elliptical'` indicating that the tip of the field vector is moving clockwise when looking towards the source of the field.

Polarization Ellipse

Draw the figure of an elliptically polarized field. Begin with an elliptically polarized input field (the horizontal and vertical components differ in magnitude and in phase) and choose the phase difference to be 60 degrees.

```
fv = [2*exp(-i*pi/3);1];
polellip(fv)
```



The rotation sense is `'Left Elliptical'` as shown by the direction of the arrow on the ellipse. The filled circle at the origin indicates that the observer is looking towards the source of the field.

Input Argument

`fv` — Field vector in linear component representation
1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix

Field vector in linear component representation specified as a 1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix. Each column contains an instance of a field specification. If `fv` is a matrix, each column in `fv` represents a field in the form of `[Eh;Ev]`, where `Eh` and `Ev` are the field's linear horizontal and vertical polarization components. If `fv` is a row vector, then the row contains the ratio of the vertical to horizontal components of the field `Ev/Eh`. For a row vector, the value `Inf` is allowed to designate the case when the ratio is computed for `Eh = 0`. `Eh` and `Ev` cannot both be set to zero.

Example: `[1;-i]`

Example: `2 + pi/3*i`

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`tau` — Tilt angle of polarization ellipse
1-by-*N* real-valued row vector

Tilt angle of polarization ellipse returned as a 1-by-*N* real-valued row vector. Each entry in `tau` contains the tilt angle of the polarization ellipse associated with each column of the field `fv`. The tilt angle is the angle between the semi-major axis of the ellipse and the horizontal axis (i.e. *x*axis) and lies in the range `[-90,90]°`.

`epsilon` — Ellipticity angle of the polarization ellipse
1-by-*N* real-valued row vector

Ellipticity angle of the polarization ellipse returned as 1-by-*N* real-valued row vector. Each entry in `epsilon` contains the ellipticity angle of the polarization ellipse associated with each column of the field `fv`. The ellipticity angle describes the shape of the ellipse and lies in the range `[-45°,45°]`.

`ar` — Axial ratio of the polarization ellipse
1-by-*N* real-valued row vector

Axial ratio of the polarization ellipse returned as a 1-by-*N* real-valued row vector. Each entry in `ar` contains the axial ratio of the polarization ellipse associated with each column of the field `fv`. The axial ratio is the signed ratio of the major-axis length to the minor-axis length of the polarization ellipse. Its absolute value is always greater than or equal to one. The sign of `ar` carries the rotational sense of the vector – a negative sign denotes left-handed rotation and a positive sign denotes right-handed rotation.

`rs` — Rotation sense of the polarization ellipse
1-by-*N* cell array of character vectors

Rotation sense of the polarization ellipse returned as a 1-by-*N* cell array of character vectors. Each entry in `rs` contains the rotation sense of the polarization ellipse associated with each column of the field `fv`. The rotation sense can be one of `'Linear'`, `'Left Circular'`, `'Right Circular'`, `'Left Elliptical'` or `'Right Elliptical'`.

Version History

Introduced in R2013a

References

[1] Mott, H., *Antennas for Radar and Communications*, John Wiley & Sons, 1992.

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`circpol2pol` | `pol2circpol` | `polratio` | `stokes`

polloss

Polarization loss

Syntax

```
rho = polloss(fv_tr,fv_rcv)
rho = polloss(fv_tr,fv_rcv,pos_rcv)
rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv)
rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv,pos_tr)
rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv,pos_tr,axes_tr)
```

Description

`rho = polloss(fv_tr,fv_rcv)` returns the loss, in decibels, because of mismatch between the polarization of a transmitted field, `fv_tr`, and the polarization of the receiving antenna, `fv_rcv`. The field vector lies in a plane orthogonal to the direction of propagation from the transmitter to the receiver. The transmitted field is represented as a 2-by-1 column vector `[Eh;Ev]`. In this vector, `Eh` and `Ev` are the field's horizontal and vertical linear polarization components with respect to the transmitter's local coordinate system. The receiving antenna's polarization is specified by a 2-by-1 column vector, `fv_rcv`. You can also specify this polarization in the form of `[Eh;Ev]` with respect to the receiving antenna's local coordinate system. In this syntax, both local coordinate axes align with the global coordinate system.

`rho = polloss(fv_tr,fv_rcv,pos_rcv)` specifies, in addition, the position of the receiver. The receiver is defined as a 3-by-1 column vector, `[x;y;z]`, with respect to the global coordinate system (position units are in meters). This syntax can use any of the input arguments in the previous syntax.

`rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv)` specifies, in addition, the orthonormal axes, `axes_rcv`. These axes define the receiver's local coordinate system as a 3-by-3 matrix. The first column gives the *x*-axis of the local system with respect to the global coordinate system. The second and third columns give the *y* and *z* axes, respectively. This syntax can use any of the input arguments in the previous syntaxes.

`rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv,pos_tr)` specifies, in addition, the position of the transmitter as a 3-by-1 column vector, `[x;y;z]`, with respect to the global coordinate system (position units are in meters). This syntax can use any of the input arguments in the previous syntaxes.

`rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv,pos_tr,axes_tr)` specifies, in addition, the orthonormal axes, `axes_tr`. These axes define the transmitter's local coordinate system as a 3-by-3 matrix. The first column gives the *x*-axis of the local system with respect to the global coordinate system. The second and third columns give the *y* and *z* axes, respectively. This syntax can use any of the input arguments in the previous syntaxes.

Examples

Mismatch Between 45° Polarized Field and Horizontally Polarized Receiver

Begin with a 45° polarized transmitted field and a receiver that is horizontally polarized. By default, the transmitter and receiver local axes coincide with the global coordinate system. Compute the polarization loss in dB.

```
fv_tr = [1;1];
fv_rcv = [1;0];
rho = polloss(fv_tr,fv_rcv)
```

```
rho = 3.0103
```

The loss is 3 dB as expected because only half the power of the field matches to the receive antenna polarization.

Polarization Loss Unaffected by Receiver Position

Begin with identical transmitter and receiver polarizations. Place the receiver at a position 100 meters along the *y*-axis. The transmitter is at the origin (its default position) and both local coordinate axes coincide with the global coordinate system (by default). First, compute the polarization loss. Then, move the receiver 100 meters along the *x*-axis, and compute the polarization loss again.

```
fv_tr = [1;0];
fv_rcv = [1;0];
pos_rcv = [0;100;0];
rho(1) = polloss(fv_tr,fv_rcv,pos_rcv);
pos_rcv = [100;100;0];
rho(2) = polloss(fv_tr,fv_rcv,pos_rcv)
```

```
rho = 1×2

     0     0
```

No polarization loss occurs at either position. The spherical basis vectors of each antenna are parallel to other antenna and the polarization vectors are the same.

Loss Due to Receiver Axes Rotation

Start with identical transmitter and receiver polarizations. Put the receiver at a position 100 meters along the *y*-axis. The transmitter is at the origin (default) and both local coordinate axes coincide with the global coordinate system (default). Compute the loss, and then rotate the receiver 30° around the *y*-axis. This rotation changes the azimuth and elevation of the transmitter with respect to the receiver and, therefore, the direction of polarization.

```
fv_tr = [1;0];
fv_rcv = [1;0];
pos_rcv = [0;100;0];
ax_rcv = azelaxes(0,0);
rho(1) = polloss(fv_tr,fv_rcv,pos_rcv,ax_rcv);
```

```
ax_rcv = roty(30)*ax_rcv;
rho(2) = polloss(fv_tr,fv_rcv,pos_rcv,ax_rcv)
```

```
rho = 1×2

         0    1.2494
```

The receiver polarization vector remains unchanged. However, rotating the local coordinate system changes the direction of the field of the receiving antenna polarization with respect to global coordinates. This change results in a 1.2 dB loss.

Polarization Loss Unaffected by Transmitter Position

Start with identical transmitter and receiver polarizations. Put the receiver at a position 100 meters along the *y*-axis. The transmitter is at the origin (default) and both local coordinate axes coincide with the global coordinate system (default). First, compute the polarization loss. Then, move the transmitter 100 meters along the *x*-axis and 100 meters along the *y*-axis, and compute the polarization loss again.

```
fv_tr = [1;0];
fv_rcv = [1;0];
pos_rcv = [0;100;0];
ax_rcv = azelaxes(0,0);
pos_tr = [0;0;0];
rho(1) = polloss(fv_tr,fv_rcv,pos_rcv,ax_rcv,pos_tr);
pos_tr = [100;100;0];
rho(2) = polloss(fv_tr,fv_rcv,pos_rcv,ax_rcv,pos_tr)
```

```
rho = 1×2

     0     0
```

There is no polarization loss at either position because the spherical basis vectors of each antenna are parallel to their counterparts and the polarization vectors are the same.

Plot Polarization Loss as Receiving Antenna Rotates

Specifying identical transmitter and receiver polarizations, plot the loss as the local receiving antenna axes rotate around the *x*-axis.

```
fv_tr = [1;0];
fv_rcv = [1;0];
```

The position of the transmitting antenna is at the origin and its local axes align with the global coordinate system. The position of the receiving antenna is 100 meters along the global *x*-axis. However, its local *x*-axis points towards the transmitting antenna.

```
pos_tr = [0;0;0];
axes_tr = azelaxes(0,0);
```

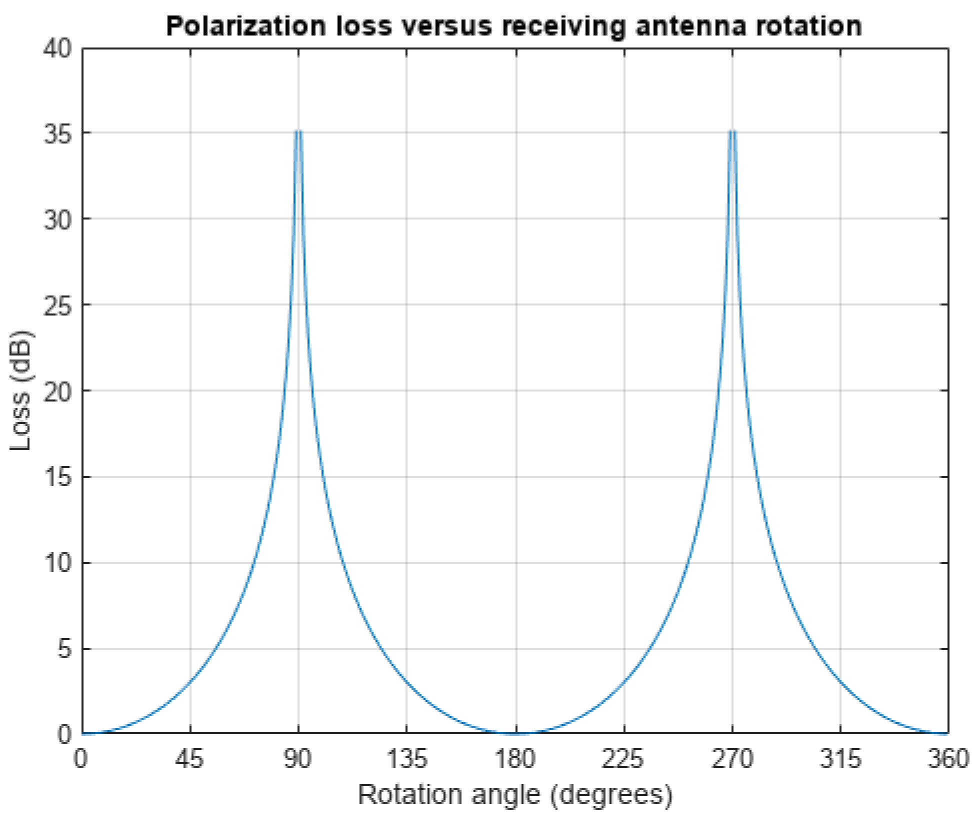
```
pos_rcv = [100;0;0];
axes_rcv0 = rotz(180)*azelaxes(0,0);
```

Rotate the receiving antenna around its local *x*-axis in one-degree increments. Compute the loss for each angle.

```
angles = [0:1:359];
n = size(angles,2);
rho = zeros(1,n); % Initialize space
for k = 1:n
    axes_rcv = rotx(angles(k))*axes_rcv0;
    rho(k) = polloss(fv_tr,fv_rcv,pos_tr,axes_tr,...
        pos_rcv,axes_rcv);
end
```

Plot the polarization loss.

```
hp = plot(angles,rho);
hax = hp.Parent;
hax.XLim = [0,360];
xticks = (0:(n-1))*45;
hax.XTick = xticks;
grid;
title('Polarization loss versus receiving antenna rotation')
xlabel('Rotation angle (degrees)');
ylabel('Loss (dB)');
```



The angle-loss plot shows nulls (`Inf` dB) at 90 degrees and 270 degrees where the polarizations are orthogonal.

Input Arguments

`fv_tr` — Transmitted field vector in linear component representation
2-by-1 complex-valued column vector

The transmitted field vector in linear component representation specified as a 2-by-1, complex-valued column vector `[Eh;Ev]`. In this vector, `Eh` and `Ev` are the field's horizontal and vertical linear components.

Example: [1;1]

Data Types: `double`
Complex Number Support: Yes

`fv_rcv` — Receiver polarization vector in linear component representation
2-by-1 complex-valued column vector

Receiver polarization vector in linear component representation specified as a 2-by-1, complex-valued column vector `[Eh;Ev]`. In this vector, `Eh` and `Ev` are the polarization vector's horizontal and vertical linear components.

Example: [0;1]

Data Types: `double`
Complex Number Support: Yes

`pos_rcv` — Receiving antenna position
[0;0;0] (default) | 3-by-1 real-valued column vector

Receiving antenna position specified as a 3-by-1, real-valued column vector. The components of `pos_rcv` are specified in the global coordinate system as `[x;y;z]`.

Example: [1000;0;0]

Data Types: `double`

`axes_rcv` — Receiving antenna local coordinate axes
3-by-3 identity matrix (default) | 3-by-3 real-valued matrix

Receiving antenna local coordinate axes specified as a 3-by-3, real-valued matrix. Each column is a unit vector specifying the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively, with respect to the global coordinate system. Each column is written in `[x;y;z]` form. If `axes_rcv` is specified as the identity matrix, the local coordinate system is aligned with the global coordinate system.

Example: [1, 0, 0; 0, 1, 0; 0, 0 ,1]

Data Types: `double`

`pos_tr` — Transmitter position
[0;0;0] (default) | 3-by-1 real-valued column vector

Transmitter position specified as a 3-by-1, real-valued column vector. The components of `pos_tr` are specified in the global coordinate system as `[x;y;z]`.

Example: [0;0;0]

Data Types: `double`

`axes_tr` — Transmitting antenna local coordinate axes
3-by-3 identity matrix (default) | 3-by-3 real-valued matrix

Transmitting antenna local coordinate axes specified as a 3-by-3, real-valued matrix. Each column is a unit vector specifying the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively, with respect to the global coordinate system. Each column is written in `[x;y;z]` form. If `axes_tr` is the identity matrix, the local coordinate system is aligned with the global coordinate system.

Example: [1, 0, 0; 0, 1, 0; 0, 0 ,1]

Data Types: `double`

Output Arguments

`rho` — Polarization loss
scalar

Polarization loss returned as scalar in decibel units. The polarization loss is the projection of the normalized transmitted field vector into the normalized receiving antenna polarization vector. Its value lies between zero and unity. When converted into dB, (and a sign changed to show loss as positive) its value lies between 0 and `-Inf`.

More About

Polarization Loss Due to Field and Receiver Mismatch

Loss occurs when a receiver is not matched to the polarization of an incident electromagnetic field.

In the case of the polarization of a field emitted by a transmitting antenna, first, look at the far zone of the transmitting antenna, as shown in the following figure. At this location—which is the location of the receiving antenna—the electromagnetic field is orthogonal to the direction from transmitter to receiver.

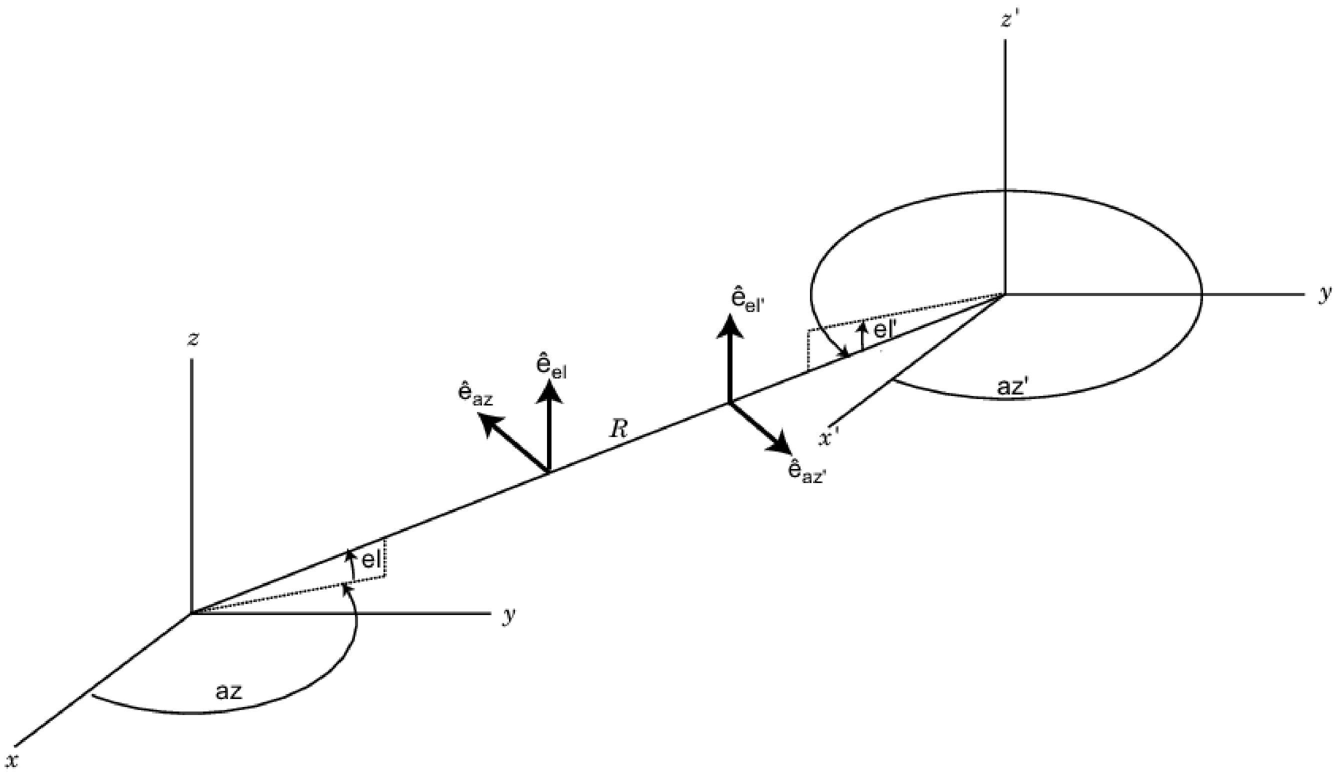
You can represent the transmitted electromagnetic field, `fv_tr`, by the components of a vector with respect to a spherical basis of the transmitter’s local coordinate system. The orientation of this basis depends on its direction from the origin. The direction is specified by the azimuth and elevation of the receiving antenna with respect to the transmitter’s local coordinate system. Then, the transmitter’s polarization, in terms of the spherical basis vectors of the transmitter’s local coordinate system, is

$$\mathbf{E} = E_H\widehat{\mathbf{e}}_{az} + E_V\widehat{\mathbf{e}}_{el} = E_m\mathbf{P}_i$$

In the same manner, the receiver’s polarization vector, `fv_rcv`, is defined with respect to a spherical basis in the receiver’s local coordinate system. Now, the azimuth and elevation specify the transmitter’s position with respect to the receiver’s local coordinate system. You can write the receiving antennas polarization in terms of the spherical basis vectors of the receiver’s local coordinate system:

$$\mathbf{P} = P_H\widehat{\mathbf{e}}'_{az} + P_V\widehat{\mathbf{e}}'_{el}$$

This figure shows the construction of the different transmitter and receiver local coordinate systems. It also shows the spherical basis vectors with which to write the field components.



The polarization loss is the projection (or dot product) of the normalized transmitted field vector onto the normalized receiver polarization vector. Notice that the loss occurs because of the mismatch in direction of the two vectors not in their magnitudes. Because the vectors are defined in different coordinate systems, they must be converted to the global coordinate system in order to form the projection. The polarization loss is defined by:

$$\rho = \frac{|\mathbf{E}_i \cdot \mathbf{P}|^2}{|\mathbf{E}_i|^2\,|\mathbf{P}|^2}$$

Version History

Introduced in R2013a

References

[1] Mott, H. *Antennas for Radar and Communications*.John Wiley & Sons, 1992.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`polellip` | `stokes`

polratio

Ratio of vertical to horizontal linear polarization components of a field

Syntax

```
p = polratio(fv)
```

Description

`p = polratio(fv)` returns the ratio of the vertical to horizontal component of the field or set of fields contained in `fv`.

Each column of `fv` contains the linear polarization components of a field in the form `[Eh;Ev]`, where `Eh` and `Ev` are the field's linear horizontal and vertical polarization components. The expression of a field in terms of a two-row vector of linear polarization components is called the Jones vector formalism. The argument `fv` can refer to either the electric or magnetic part of an electromagnetic wave.

Each entry in `p` contains the ratio `Ev/Eh` of the components of `fv`.

Examples

Polarization Ratio for 45° Linearly Polarized Field

Determine the polarization ratio for a linearly polarized field (when the horizontal and vertical components of a field have the same phase).

```
fv = [2;2];
p = polratio(fv)
```

```
p = 1
```

The polarization ratio is real. Because the components have equal amplitudes, the polarization ratio is unity.

Polarization Ratios for Two Fields

Compute the polarization ratios for two fields. The first field is *(2;i)* and the second is *(i;1)*.

```
fv = [2,1i;1i,1];
p = polratio(fv)
```

```
p = 1×2 complex

   0.0000 + 0.5000i   0.0000 - 1.0000i
```

Polarization Ratio for Vertically Polarized Field

Determine the polarization ratio for a vertically polarized field (the horizontal component of the field vanishes).

```
fv = [0;2];
p = polratio(fv)
```

```
p = Inf
```

The polarization ratio is infinite as expected from the definition, *Ev/Eh*.

Input Arguments

`fv` — Field vector in linear component representation
2-by-*N* complex-valued matrix

Field vector in linear component representation specified as a 2-by-*N* complex-valued matrix. Each column of `fv` contains an instance of a field specified by `[Eh;Ev]`, where `Eh` and `Ev` are the field's linear horizontal and vertical polarization components. Two rows of the same column cannot both be zero.

Example: [2 , i; i, 1]

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`p` — Polarization ratio
1-by-*N* complex-valued row vector

Polarization ratio returned as a 1-by-*N* complex-valued row vector. `p` contains the ratio of the components of the second row of `fv` to the first row, `Ev/Eh`.

Version History
Introduced in R2013a

References

[1] Mott, H., *Antennas for Radar and Communications*, John Wiley & Sons, 1992.

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`circpol2pol` | `pol2circpol` | `polellip` | `stokes`

polsignature

Copolarization and cross-polarization signatures

Syntax

```
resp = polsignature(rcsmat)
resp = polsignature(rcsmat,type)
resp = polsignature(rcsmat,type,epsilon)
resp = polsignature(rcsmat,type,epsilon,tau)

polsignature( ___ )
```

Description

`resp = polsignature(rcsmat)` returns the normalized radar cross-section copolarization (co-pol) signature, `resp` (in square meters), determined from the scattering cross section matrix, `rcsmat` of an object. The signature is a function of the transmitting antenna polarization, specified by the ellipticity angle and the tilt angle of the polarization ellipse. In this syntax case, the ellipticity angle takes the values `[-45:45]` and the tilt angle takes the values `[-90:90]`. The output `resp` is a 181-by-91 matrix whose elements correspond to the signature at each ellipticity angle-tilt angle pair.

`resp = polsignature(rcsmat,type)`, in addition, specifies the polarization signature type as one of `'c'|'x'`, where `'c'` creates the copolarization signature and `'x'` creates the cross-polarization (cross-pol) signature. The default value of this parameter is `'c'`. The output `resp` is a 181-by-91 matrix whose elements correspond to the signature at each ellipticity angle-tilt angle pair. This syntax can use the input arguments in the previous syntax.

`resp = polsignature(rcsmat,type,epsilon)`, in addition, specifies the transmit antenna polarization's ellipticity angle (in degrees) as a length-*M* vector. The angle `epsilon` must lie between –45° and 45°. The argument `resp` is a 181-by-*M* matrix whose elements correspond to the signature at each ellipticity angle-tilt angle pair. This syntax can use any of the input arguments in the previous syntaxes.

`resp = polsignature(rcsmat,type,epsilon,tau)`, in addition, specifies the tilt angle of the polarization ellipse of the transmitted wave (in degrees) as a length-*N* vector. The angle `tau` must be between –90° and 90°. The signature, `resp`, is represented as a function of the transmitting antenna polarization. The transmitting antenna polarization is characterized by the ellipticity angle, `epsilon`, and the tilt angle, `tau`. The argument `resp` is a *N*-by-*M* matrix whose elements correspond to the signature at each ellipticity angle-tilt angle pair. This syntax can use any of the input arguments in the previous syntaxes.

`polsignature( ___ )` plots a three dimensional surface using any of the syntax forms specified above.

Examples

Copolarization Signature of a Dihedral

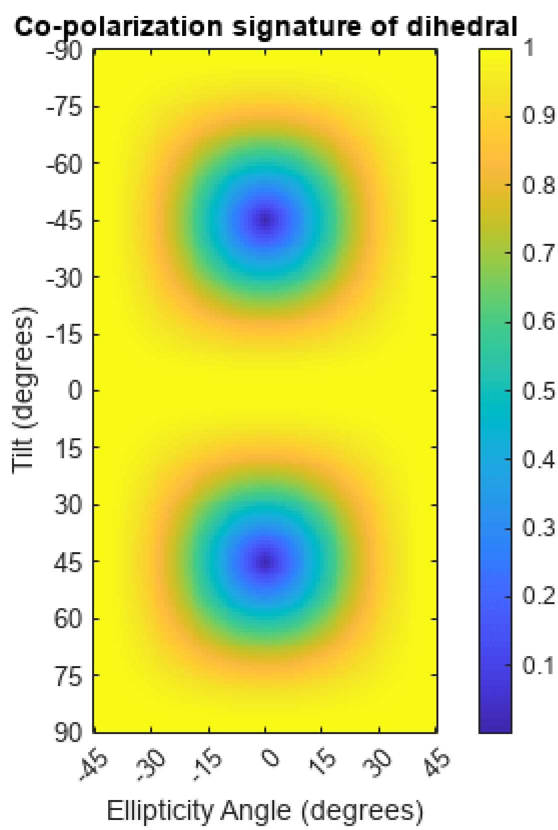
Calculate and plot the copolarization response to the scattering cross-section matrix, `rscmat`, of a dihedral object. Specify the ellipticity angle values as `[-45:45]` and the tilt angle values as `[-90:90]`. Display the response matrix as an image.

Calculate the copolarization response.

```
rscmat = [-1,0;0,1];
resp = polsignature(rscmat);
```

Plot the copolarization response.

```
el = [-45:45];
tilt = [-90:90];
imagesc(el,tilt,resp);
ylabel('Tilt (degrees)');
xlabel('Ellipticity Angle (degrees)')
axis image
ax = gca;
ax.XTick = [-45:15:45];
ax.YTick = [-90:15:90];
title('Co-polarization signature of dihedral');
colorbar;
```



Cross-Polarization Signature of a Dihedral

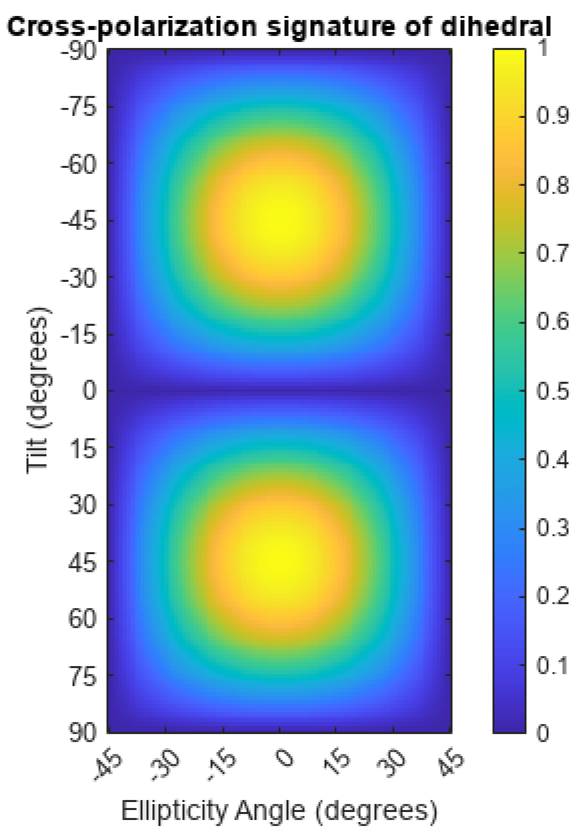
Calculate and plot the cross-polarization response to the scattering cross-section matrix, `rscmat`, of a dihedral object. Specify the ellipticity angle values as [-45:45] and the tilt angle values as [-90:90]. Display the response matrix as an image.

Calculate the cross-polarization response. To do this, set the `type` argument to `'x'`.

```
rscmat = [-1,0;0,1];
resp = polsignature(rscmat,'x');
```

Plot the cross-polarization response.

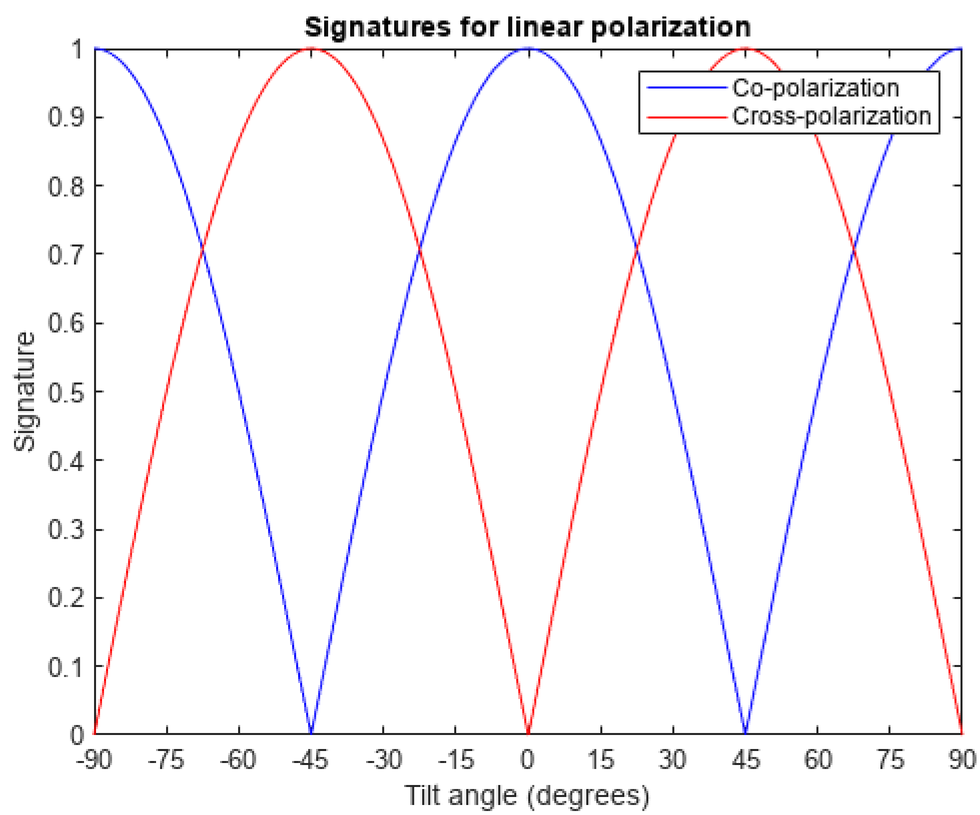
```
el = [-45:45];
tilt = [-90:90];
imagesc(el,tilt,resp);
ylabel('Tilt (degrees)');
xlabel('Ellipticity Angle (degrees)');
axis image
ax = gca;
ax.XTick = [-45:15:45];
ax.YTick = [-90:15:90];
title('Cross-polarization signature of dihedral');
colorbar;
```



Signatures for Linear Polarization with Varied Tilt Angles

Set the ellipticity angle to zero, and vary the tilt angle from -90 to +90 degrees to generate all possible linear polarization directions. Then, plot both the copolarization and cross-polarization signatures.

```
rscmat = [-1,0;0,1];
el = [0];
respc = polsignature(rscmat,'c',el);
respx = polsignature(rscmat,'x',el);
tilt = [-90:90];
plot(tilt,respc,'b',tilt,respx,'r')
ax = gca;
ax.XLim = [-90,90];
ax.XTick = [-90:15:90];
legend('Co-polarization','Cross-polarization')
title('Signatures for linear polarization')
xlabel('Tilt angle (degrees)')
ylabel('Signature')
```



Copolarization Signature of Dihedral for Left and Right Circular Polarizations

This example shows how to obtain numerical values for the polarization signatures of a dihedral target for left and right circularly polarized incident waves.

Specify the radar cross-section matrix of a dihedral

```
rscmat = [-1,0;0,1];
```

Specify a left circularly-polarized wave and obtain its tilt angle and ellipticity.

```
fv = 1/sqrt(2)*[1;1i];
[tilt_lcp,el_lcp] = polellip(fv);
disp([tilt_lcp,el_lcp])
```

```
    45    45
```

Specify a right circularly-polarized wave by complex conjugation of a left circularly-polarized wave. Obtain the polarization ellipse tilt angle and ellipticity.

```
[tilt_rcp,el_rcp] = polellip(conj(fv));
disp([tilt_rcp,el_rcp])
```

```
    45   -45
```

Both tilt angles are 45 degrees. Compute the copolarization and cross-polarization signatures for the two waves.

```
el = [el_lcp, el_rcp];
tilt = tilt_rcp;
respc = polsignature(rscmat,'c',el,tilt);
respx = polsignature(rscmat,'x',el,tilt);
disp(respc)
```

```
     1     1
```

```
disp(respx)
```

```
     1     1
```

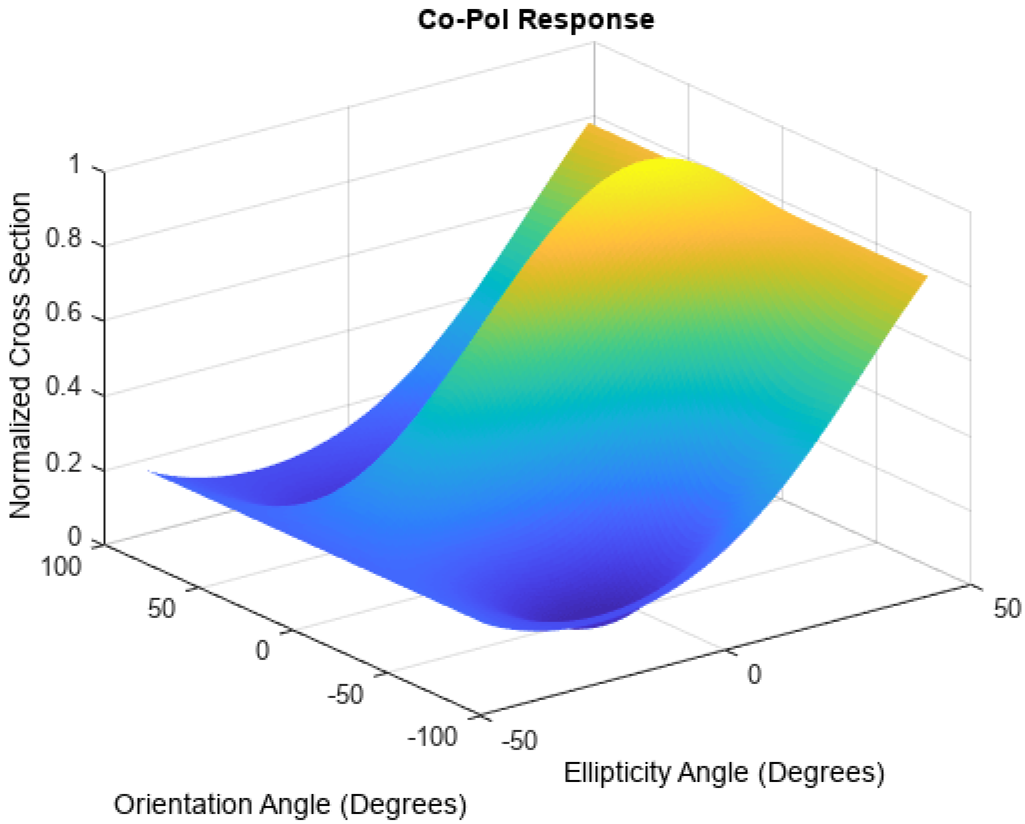
Surface Plot of Copolarization Signature of General Target

Use a general RCSM matrix to create a 3-D surface plot.

```
rscmat = [1i*2,0.5; 0.5, -1i];
el = [-45:45];
tilt = [-90:90];
```

With no output arguments, `polsignature` automatically creates a surface plot.

```
polsignature(rscmat,'c',el,tilt);
```



Input Arguments

`rcsmat` — Radar cross-section scattering matrix
2-by-2 complex-valued matrix

Radar cross-section scattering matrix (RCSM) of an object specified as a 2-by-2, complex-valued matrix. The radar cross-section scattering matrix describes the polarization of a scattered wave as a function of the polarization of an incident wave upon a target. The response to an incident wave can be construct from the individual responses to the incident field's horizontal and vertical polarization components. These components are taken with respect to the transmit antenna or array local coordinate system. The scattered wave can be decomposed into horizontal and vertical polarization components with respect to the receive antenna or array local coordinate system. The matrix *RCSM* contains four components `[rcs_hh rcs_hv;rcs_vh rcs_vv]` where each component is the radar cross section defined by the polarization of the transmit and receive antennas.

- `rcs_hh` – Horizontal response due to horizontal transmit polarization component
- `rcs_hv` – Horizontal response due to vertical transmit polarization component
- `rcs_vh` – Vertical response due to horizontal transmit polarization component
- `rcs_vv` – Vertical response due to vertical transmit polarization component

In the monostatic radar case, when the wave is backscattered, the RCSM matrix is symmetric.

Example: `[-1,1i;1i,1]`

Data Types: `double`
Complex Number Support: Yes

`type` — Polarization signature type
`'c'` (default) | single character `'c'`|`'x'`

Polarization signature type of the scattered wave specified by a single character: `'c'` denoting the copolarized signature or `'x'` denoting the cross-polarized signature.

Example: `'x'`

Data Types: `char`

`epsilon` — Ellipticity angle of the polarization ellipse of the transmitted wave
`[-45:45]` (default) | scalar or 1-by-*M* real-valued row vector

Ellipticity angle of the polarization ellipse of the transmitted wave specified as a length-*M* vector. Units are degrees. The ellipticity angle describes the shape of the ellipse. By definition, the tangent of the ellipticity angle is the signed ratio of the semiminor axis to semimajor axis of the polarization ellipse. Since the absolute value of this ratio cannot exceed unity, the ellipticity angle lies between ±45°.

Example: `[-45:0.5:45]`

Data Types: `double`

`tau` — Tilt angle of the polarization ellipse of the transmitted wave
`[-90:90]` (default) | scalar or 1-by-*N* real-valued row vector.

Tilt angle of the polarization ellipse of the transmitted wave specified as a length-*N* vector. Units are degrees. The tilt angle is defined as the angle between the semimajor axis of the ellipse and the *x*-axis. Because the ellipse is symmetrical, an ellipse with a tilt angle of 100° is the same ellipse as one with a tilt angle of –80°. Therefore, the tilt angle need only be specified between ±90°.

Example: `[-30:2:30]`

Data Types: `double`

Output Arguments

`resp` — Normalized magnitude response
scalar or *N*-by-*M* real-valued matrix.

Normalized magnitude response returned as a scalar or *N*-by-*M*, real-valued matrix having values between 0 and 1. `resp` returns a value for each ellipticity-tilt angle pair.

More About

Scattering Cross-Section Matrix

Scattering cross-section matrix determines response of an object to incident polarized electromagnetic field.

When a polarized plane wave is incident on an object, the amplitude and polarization of the scattered wave may change with respect to the incident wave polarization. The polarization may depend upon the direction from which the scattered wave is observed. The exact way that the polarization changes depends upon the properties of the scattering object. The quantity describing the response of an object to the incident field is called the scattering cross-section matrix, S. The scattering matrix can be measured as follows: when a unit amplitude horizontally polarized wave is scattered, both a horizontal and vertical scattered component are produced. Call these two components S_{HH} and S_{VH}. These are complex numbers containing the amplitude and phase changes from the incident wave. Similarly, when a unit amplitude vertically polarized wave is scattered, the horizontal and vertical scattered component produced are S_{HV} and S_{VV}. Because any incident field can be decomposed into horizontal and vertical components, stack these quantities into a matrix and write the scattered field in terms of the incident field

$$\begin{bmatrix} E_H^{(sc)} \\ E_V^{(sc)} \end{bmatrix} = \begin{bmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{bmatrix} \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix} = S \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

The scattering cross section matrix depends upon the angles that the incident and scattered fields make with the object. When the incident field is backscattered to the transmitting antenna, the scattering matrix is symmetric.

Polarization Signature

Polarization signature for visualizing scattering cross-section matrix.

To understand how the scattered wave depends upon the polarization of the incident wave, an examination of all possible scattered field polarizations for each incident polarization is required. Because this amount of data is difficult to visualize, you can look at two particular scattered polarizations:

- Choose one polarization that has the same polarization as the incident field (copolarization)
- Choose a second one that is orthogonal to the polarization of the incident field (cross-polarization)

Both the incident and orthogonal polarization states can be specified in terms of the tilt angle-ellipticity angle pair (τ, ε). From the incident field tilt and ellipticity angles, the unit incident polarization vector can be expressed as

$$\begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix} = \begin{bmatrix} \cos\tau & -\sin\tau \\ \sin\tau & \cos\tau \end{bmatrix} \begin{bmatrix} \cos\varepsilon \\ j\sin\varepsilon \end{bmatrix}$$

while the orthogonal polarization vector is

$$\begin{bmatrix} E_H^{(inc)\perp} \\ E_V^{(inc)\perp} \end{bmatrix} = \begin{bmatrix} -\sin\tau & -\cos\tau \\ \cos\tau & -\sin\tau \end{bmatrix} \begin{bmatrix} \cos\varepsilon \\ -j\sin\varepsilon \end{bmatrix}$$

To form the copolarization signature, use the RCSM matrix, S, to compute:

$$P^{(co)} = \begin{bmatrix} E_H^{(inc)} & E_V^{(inc)} \end{bmatrix}^* S \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

where $[\,]^*$ denotes complex conjugation. For the cross-polarization signature, compute

$$P^{(cross)} = \begin{bmatrix} E_H^{(inc)\perp} & E_V^{(inc)\perp} \end{bmatrix}^* S \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

The output of this function is the absolute value of each signature normalized by its maximum value.

Version History

Introduced in R2013a

References

[1] Mott, H. *Antennas for Radar and Communications*.John Wiley & Sons, 1992.

[2] Fawwaz, U. and C. Elachi. *Radar Polarimetry for Geoscience Applications*. Artech House, 1990.

[3] Lee, J. and E. Pottier. *Polarimetric Radar Imaging: From Basics to Applications*. CRC Press, 2009.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.
- Supported only when output arguments are specified.

See Also

`polellip` | `polloss` | `stokes`

pulsint

Pulse integration

Syntax

```
Y = pulsint(X)
Y = pulsint(X,METHOD)
```

Description

`Y = pulsint(X)` performs video (noncoherent) integration of the pulses in `X` and returns the integrated output in `Y`. Each column of `X` is one pulse.

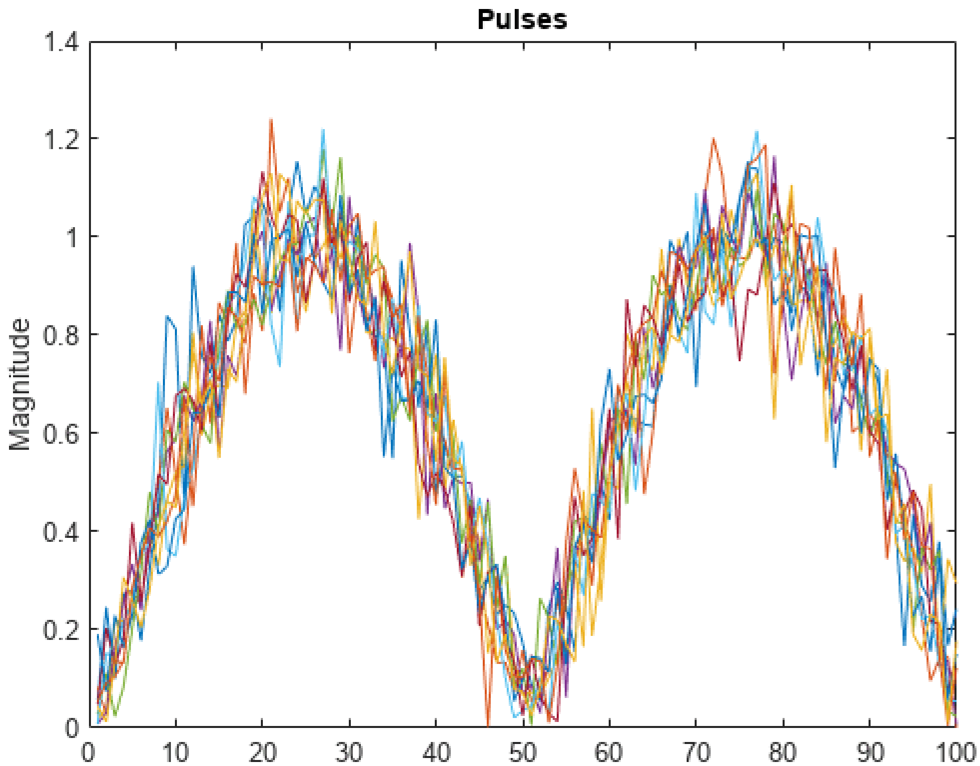
`Y = pulsint(X,METHOD)` performs pulse integration using the specified method. `METHOD` is `"coherent"` or `"noncoherent"`.

Examples

Noncoherent and Coherent Integration of Pulses

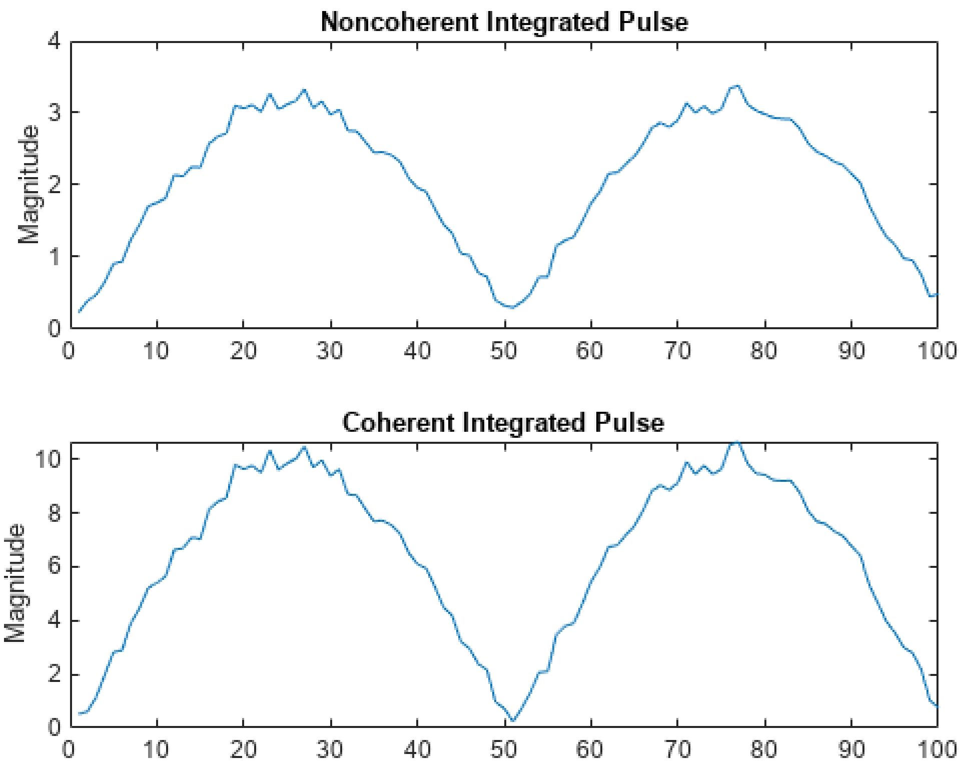
Generate ten pulses of a sinusoid with added Gaussian white noise. Plot the pulse magnitudes.

```
npulse = 10;
x = repmat(sin(2*pi*(0:99)'/100),1,npulse) + 0.1*randn(100,npulse);

plot(abs(x))
ylabel("Magnitude")
title("Pulses")
```



Perform noncoherent integration of the pulses. Repeat the computation, but now perform coherent integration. Plot the magnitudes of the integrated pulses

```
yn = pulsint(x);

subplot(2,1,1)
plot(abs(yn))
ylabel("Magnitude")
title("Noncoherent Integrated Pulse")

yc = pulsint(x,"coherent");

subplot(2,1,2)
plot(abs(yc))
ylabel("Magnitude")
title("Coherent Integrated Pulse")
```



Input Arguments

X — Pulse input data
matrix

Pulse input data, specified as a matrix. Each column of X is one pulse.

Example: `sin(pi./[32 32].*(0:2047)')+randn(2048,2)/10`

Data Types: `double`

`METHOD` — Pulse integration method
`"noncoherent"` (default) | `"coherent"`

Pulse integration method, specified as `"coherent"` or `"noncoherent"`. `METHOD` is the method used to integrate the pulses in the columns of X. The values are not case sensitive.

Data Types: `char` | `string`

Output Arguments

Y — Integrated pulse
column vector

Integrated pulse, returned as an N-by-1 column vector, where N is the number of rows in the input X.

More About

Coherent Integration

Let X_{ij} denote the *(i,j)*-th entry of an *M*-by-*N* matrix of pulses *X*.

The coherent integration of the pulses in *X* is:

$$Y_i = \sum_{j=1}^{N} X_{ij}$$

Noncoherent (video) Integration

Let X_{ij} denote the *(i,j)*-th entry of an *M*-by-*N* matrix of pulses *X*.

The noncoherent (video) integration of the pulses in *X* is:

$$Y_i = \sqrt{\sum_{j=1}^{N} \left|X_{ij}\right|^2}$$

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

See Also

`phased.MatchedFilter`

radialspeed

Relative radial speed

Syntax

```
Rspeed = radialspeed(Pos,V)
Rspeed = radialspeed(Pos,V,RefPos)
Rspeed = radialspeed(Pos,V,RefPos,RefV)
```

Description

`Rspeed = radialspeed(Pos,V)` returns the radial speed of the given platforms relative to a reference platform. The platforms have positions `Pos` and velocities `V`. The reference platform is stationary and is located at the origin.

`Rspeed = radialspeed(Pos,V,RefPos)` specifies the position of the reference platform.

`Rspeed = radialspeed(Pos,V,RefPos,RefV)` specifies the velocity of the reference platform.

Examples

Radial Speed of Target Relative to Stationary Platform

Calculate the radial speed of a target relative to a stationary platform. Assume the target is located at *(20,20,0)* meters in Cartesian coordinates and is moving with velocity *(10,10,0)* meters per second. The reference platform is located at *(1,1,0)*.

```
rspeed = radialspeed([20; 20; 0],[10; 10; 0],[1; 1; 0])
```

```
rspeed = -14.1421
```

Negative radial speed indicates that the target is receding from the platform.

Input Arguments

Pos — Positions of platforms
matrix

Positions of platforms, specified as a 3-by-N matrix. Each column specifies a position in the form [*x*; *y*; *z*], in meters.

Example: `[20; 20; 0]`

Data Types: `double`

V — Velocities of platforms
matrix

Velocities of platforms, specified as a 3-by-N matrix. Each column specifies a velocity in the form [*x*; *y*; *z*], in meters per second.

Example: `[10; 10; 0]`

Data Types: `double`

`RefPos` — Position of reference platform

`[0; 0; 0]` (default) | vector

Position of reference platform, specified as a 3-by-1 vector. The vector has the form [*x*; *y*; *z*], in meters.

Example: `[1; 1; 0]`

Data Types: `double`

`RefV` — Velocity of reference platform

`[0; 0; 0]` (default) | vector

Velocity of reference platform, specified as a 3-by-1 vector. The vector has the form [*x*; *y*; *z*], in meters per second.

Example: `[1; -1; 0]`

Data Types: `double`

Output Arguments

`Rspeed` — Radial speed

vector

Radial speed in meters per second, returned as an N-by-1 vector. Each number in the vector represents the radial speed of the corresponding platform. Positive numbers indicate that the platform is approaching the reference platform. Negative numbers indicate that the platform is moving away from the reference platform.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phased.Platform` | `speed2dop`

Topics

"Doppler Shift and Pulse-Doppler Processing"

"Motion Modeling in Phased Array Systems"

rainpl

RF signal attenuation due to rainfall

Syntax

```
L = rainpl(range,freq,rainrate)
L = rainpl(range,freq,rainrate,elev)
L = rainpl(range,freq,rainrate,elev,tau)
L = rainpl(range,freq,rainrate,elev,tau,pct)
```

Description

`L = rainpl(range,freq,rainrate)` returns the signal attenuation, `L`, due to rainfall. In this syntax, attenuation is a function of signal path length, `range`, signal frequency, `freq`, and rain rate, `rainrate`. The path elevation angle and polarization tilt angles are assumed to zero.

The `rainpl` function applies the International Telecommunication Union (ITU) rainfall attenuation model to calculate path loss of signals propagating in a region of rainfall [1]. The function applies when the signal path is contained entirely in a uniform rainfall environment. Rain rate does not vary along the signal path. The attenuation model applies only for frequencies at 1–1000 GHz.

`L = rainpl(range,freq,rainrate,elev)` also specifies the elevation angle, `elev`, of the propagation path.

`L = rainpl(range,freq,rainrate,elev,tau)` also specifies the polarization tilt angle, `tau`, of the signal.

`L = rainpl(range,freq,rainrate,elev,tau,pct)` also specifies the specified percentage of time, `pct`. `pct` is a scalar in the range of 0.001–1, inclusive. The attenuation, `L`, is computed from a power law using the long-term statistical 0.01% rain rate (in mm/h).

Examples

Signal Attenuation Due to Rainfall

Compute the signal attenuation due to rainfall for a 20 GHz signal over a distance of 10 km in light and heavy rain.

Propagate the signal in a light rainfall of 1 mm/hr.

```
rr = 1.0;
L = rainpl(10000,20.0e9,rr)
```

```
L = 1.3009
```

Propagate the signal in a heavy rainfall of 10 mm/hr.

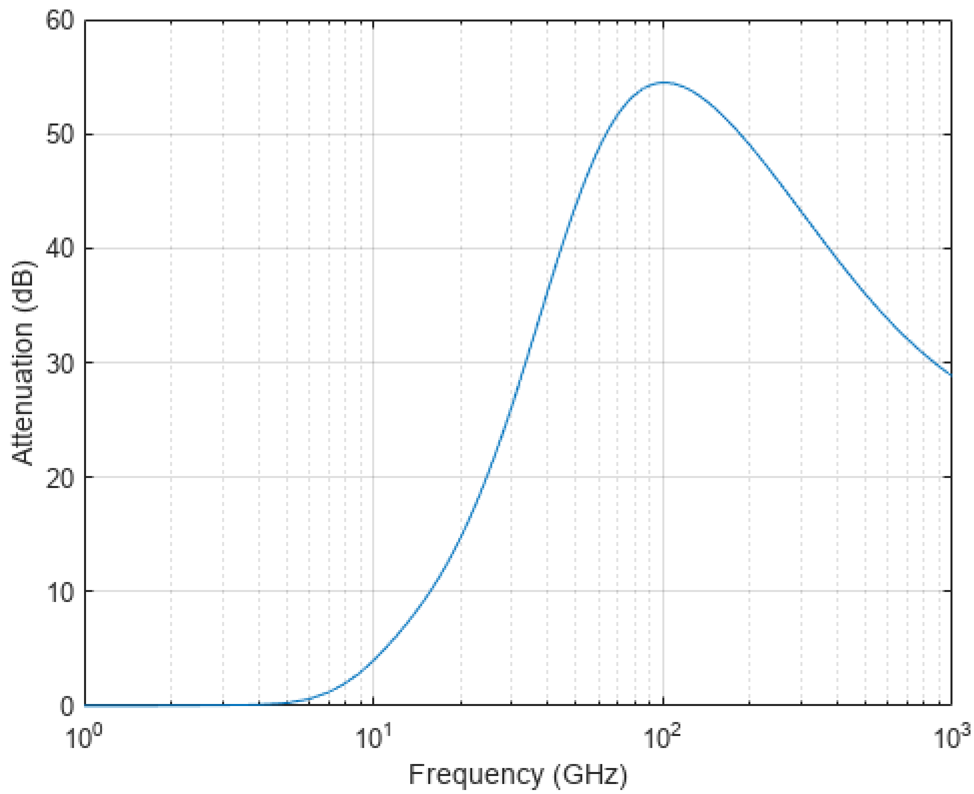
```
rr = 10.0;
L = rainpl(10000,20.0e9,rr)
```

```
L = 8.1584
```

Signal Attenuation Due to Rainfall as Function of Frequency

Plot the signal attenuation due to a 20 mm/hr statistical rainfall for signals in the frequency range from 1 to 1000 GHz. The path distance is 10 km.

```
rr = 20.0;
freq = [1:1000]*1e9;
L = rainpl(10000,freq,rr);
semilogx(freq/1e9,L)
grid
xlabel('Frequency (GHz)')
ylabel('Attenuation (dB)')
```



Signal Attenuation Due to Rainfall as Function of Elevation Angle

Compute the signal attenuation due to heavy rain as a function of elevation angle. Elevation angles vary from 0 to 90 degrees. Assume a path distance of 100 km and a signal frequency of 100 GHz.

Set the rain rate to 10 mm/hr.

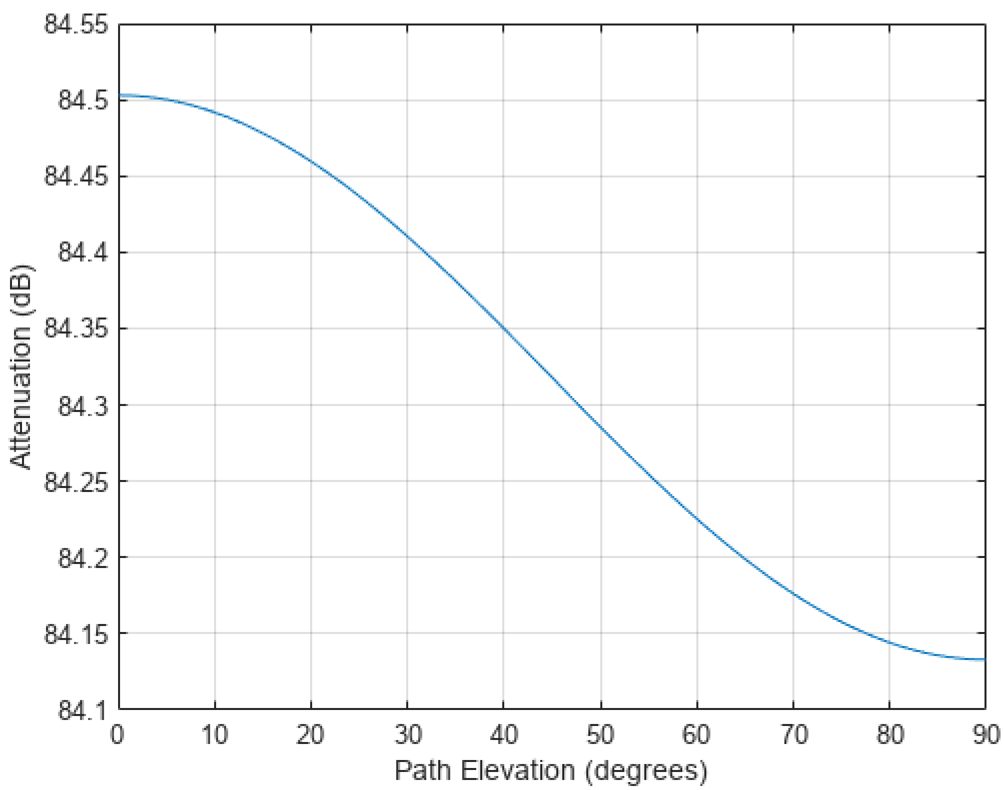
```
rr = 10.0;
```

Set the elevation angles, frequency, range.

```
elev = [0:1:90];
freq = 100.0e9;
rng = 100000.0*ones(size(elev));
```

Compute and plot the loss.

```
L = rainpl(rng,freq,rr,elev);
plot(elev,L)
grid
xlabel('Path Elevation (degrees)')
ylabel('Attenuation (dB)')
```



Signal Attenuation Due to Rainfall as Function of Polarization

Compute the signal attenuation due to heavy rainfall as a function of the polarization tilt angle. Assume a path distance of 100 km, a signal frequency of 100 GHz, and a path elevation angle of 0 degrees. Set the rainfall rate to 10 mm/hour. Plot the signal attenuation versus polarization tilt angle.

Set the polarization tilt angle to vary from -90 to 90 degrees.

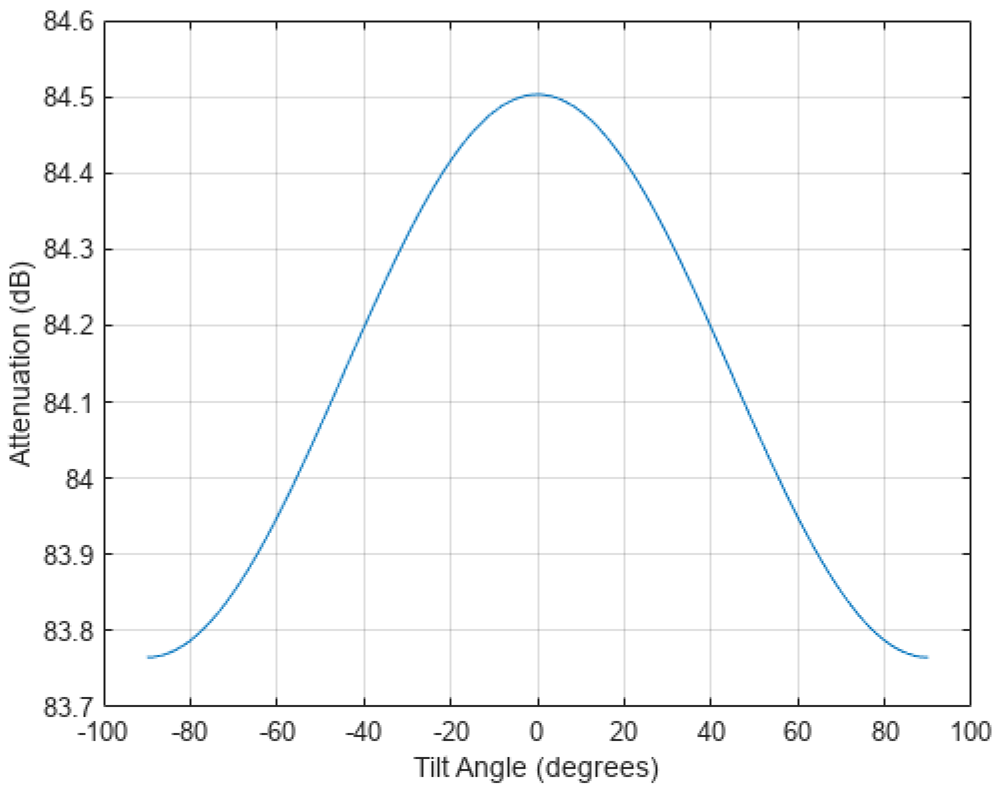
```
tau = -90:90;
```

Set the elevation angle, frequency, path distance, and rain rate.

```
elev = 0;
freq = 100.0e9;
rng = 100e3*ones(size(tau));
rr = 10.0;
```

Compute and plot the attenuation.

```
L = rainpl(rng,freq,rr,elev,tau);
plot(tau,L)
grid
xlabel('Tilt Angle (degrees)')
ylabel('Attenuation (dB)')
```



Input Arguments

range — Signal path length
nonnegative real-valued scalar | nonnegative real-valued M-by-1 column vector | nonnegative real-valued 1-by-M row vector

Signal path length, specified as a nonnegative real-valued scalar, or as a M-by-1 or 1-by-M vector. Units are in meters.

Example: `[13000.0,14000.0]`

`freq` — Signal frequency
positive real-valued scalar | nonnegative real-valued *N*-by-1 column vector | nonnegative real-valued 1-by-*N* row vector

Signal frequency, specified as a positive real-valued scalar, or as a nonnegative *N*-by-1 or 1-by-*N* vector. Frequencies must lie in the range 1–1000 GHz.

Example: `[1400.0e6,2.0e9]`

`rainrate` — Long-term statistical rain rate
nonnegative real-valued scalar

Long-term statistical rain rate, specified as a nonnegative real-valued scalar. The long-term statistical rain rate is the rain rate that is exceeded 0.01% of the time. You can adjust the percent of time using the `pct` argument. Units are in mm/hr.

Example: `1.5`

`elev` — Signal path elevation angle
0.0 (default) | real-valued scalar | real-valued *M*-by-1 column vector | real-valued 1-by-*M* row vector

Signal path elevation angle, specified as a real-valued scalar, or as an *M*-by-1 or 1-by- *M* vector. Units are in degrees between –90° and 90°. If `elev` is a scalar, all propagation paths have the same elevation angle. If `elev` is a vector, its length must match the dimension of `range` and each element in `elev` corresponds to a propagation range in `range`.

Example: `[0,45]`

`tau` — Tilt angle of polarization ellipse
0.0 (default) | real-valued scalar | real-valued *M*-by-1 column vector | real-valued 1-by-*M* row vector

Tilt angle of the signal polarization ellipse, specified as a real-valued scalar, or as an *M*-by-1 or 1-by-*M* vector. Units are in degrees between –90° and 90°. If `tau` is a scalar, all signals have the same tilt angle. If `tau` is a vector, its length must match the dimension of `range`. In that case, each element in `tau` corresponds to a propagation path in `range`.

The tilt angle is defined as the angle between the semi-major axis of the polarization ellipse and the *x*-axis. Because the ellipse is symmetrical, a tilt angle of 100° corresponds to the same polarization state as a tilt angle of -80°. Thus, the tilt angle need only be specified between ±90°.

Example: `[45,30]`

`pct` — Exceedance percentage of rainfall
`0.01` (default) | positive scalar between 0.001 and 1

Exceedance percentage of rainfall, specified as a positive scalar between 0.001 and 1. The long-term statistical rain rate is the rain rate that is exceeded `pct` of the time. Units are dimensionless.

Data Types: `double`

Output Arguments

`L` — Signal attenuation
real-valued *M*-by-*N* matrix

Signal attenuation, returned as a real-valued M-by-N matrix. Each matrix row represents a different path where M is the number of paths. Each column represents a different frequency where N is the number of frequencies. Units are in dB.

More About

Rainfall Attenuation Model

This model calculates the attenuation of signals that propagate through regions of rainfall. Rain attenuation is a dominant fading mechanism and can vary from location-to-location and from year-to-year.

Electromagnetic signals are attenuated when propagating through a region of rainfall. Rainfall attenuation is computed according to the ITU rainfall model *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. The model computes the specific attenuation (attenuation per kilometer) of a signal as a function of rainfall rate, signal frequency, polarization, and path elevation angle. The specific attenuation, γ_R, is modeled as a power law with respect to rain rate

$$\gamma_R = kR^{\alpha},$$

where R is rain rate. Units are in mm/hr. The parameter k and exponent α depend on the frequency, the polarization state, and the elevation angle of the signal path. The specific attenuation model is valid for frequencies from 1–1000 GHz.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the an effective propagation distance, d_{eff}. Then, the total attenuation is $L = d_{\mathrm{eff}}\gamma_R$.

The effective distance is the geometric distance, d, multiplied by a scale factor

$$r = \frac{1}{0.477d^{0.633}R_{0.01}^{0.073\alpha}f^{0.123} - 10.579(1 - \exp(-0.024d))}$$

where f is the frequency. The article *Recommendation ITU-R P.530-17 (12/2017): Propagation data and prediction methods required for the design of terrestrial line-of-sight systems* presents a complete discussion for computing attenuation.

The rain rate, R, used in these computations is the long-term statistical rain rate, $R_{0.01}$. This is the rain rate that is exceeded 0.01% of the time. The calculation of the statistical rain rate is discussed in *Recommendation ITU-R P.837-7 (06/2017): Characteristics of precipitation for propagation modelling*. This article also explains how to compute the attenuation for other percentages from the 0.01% value.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Version History

Introduced in R2016a

References

[1] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. 2005.

[2] Radiocommunication Sector of International Telecommunication Union. *Recommendation ITU-R P.530-17: Propagation data and prediction methods required for the design of terrestrial line-of-sight systems*. 2017.

[3] *Recommendation ITU-R P.837-7: Characteristics of precipitation for propagation modelling*

[4] Seybold, J. *Introduction to RF Propagation*. New York: Wiley & Sons, 2005.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`fspl` | `gaspl` | `fogpl` | `cranerainpl` | `LOSChannel` | `WidebandLOSChannel`

range2beat

Convert range to beat frequency

Syntax

```
fb = range2beat(r,slope)
fb = range2beat(r,slope,c)
```

Description

`fb = range2beat(r,slope)` converts the range of a dechirped linear FMCW signal to the corresponding beat frequency on page 2-341. `slope` is the slope of the FMCW sweep.

`fb = range2beat(r,slope,c)` specifies the signal propagation speed.

Examples

Maximum Beat Frequency in FMCW Radar System

Calculate the maximum beat frequency in MHz for an upsweep FMCW waveform. The waveform sweeps a 300 MHz band in 1 ms. Assume that the waveform can detect a stationary target as far as 18 km.

```
slope = 300e6/1e-3;
r = 18e3;
fb = range2beat(r,slope)/1e6
```

```
fb = 36.0249
```

Input Arguments

`r` — Range
array of nonnegative numbers

Range, specified as an array of nonnegative numbers in meters.

Data Types: `double`

`slope` — Sweep slope
nonzero scalar

Slope of FMCW sweep, specified as a nonzero scalar in hertz per second.

Data Types: `double`

`c` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar in meters per second.

Data Types: `double`

Output Arguments

`fb` — Beat frequency of dechirped signal
array of nonnegative numbers

Beat frequency of dechirped signal, returned as an array of nonnegative numbers in hertz. Each entry in `fb` is the beat frequency corresponding to the corresponding range in `r`. The dimensions of `fb` match the dimensions of `r`.

Data Types: `double`

More About

Beat Frequency

For an up-sweep or down-sweep FMCW signal, the beat frequency is $F_t - F_r$. In this expression, F_t is the transmitted signal's carrier frequency, and F_r is the received signal's carrier frequency.

For an FMCW signal with triangular sweep, the upsweep and downsweep have separate beat frequencies.

Algorithms

The function computes `2*r*slope/c`.

Version History

Introduced in R2012b

References

[1] Pace, Phillip. *Detecting and Classifying Low Probability of Intercept Radar*. Artech House, Boston, 2009.

[2] Skolnik, M.I. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`beat2range` | `dechirp` | `rdcoupling` | `stretchfreq2rng` | `phased.FMCWWaveform`

Topics
"Automotive Adaptive Cruise Control Using FMCW Technology" (Radar Toolbox)

range2bw

Convert range resolution to required bandwidth

Syntax

```
bw = range2bw(rngres)
bw = range2bw(rngres,c)
```

Description

Note The use of `range2bw` is not recommended. Use `rangeres2bw` instead.

`bw = range2bw(rngres)` returns the bandwidth needed to distinguish two targets separated by a given range. Such capability is often referred to as range resolution. The propagation is assumed to be two-way, as in a monostatic radar system.

`bw = range2bw(rngres,c)` specifies the signal propagation speed.

Input Arguments

`rngres` — Target range resolution
positive scalar | MATLAB array of positive real values

Target range resolution in meters, specified as a scalar or a MATLAB array of positive real values.

Data Types: `double`

`c` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar. Units are in meters per second.

Data Types: `double`

Output Arguments

`bw` — Required bandwidth
positive scalar | MATLAB array of positive real values

Required bandwidth in hertz, returned as a MATLAB array of positive real values. The dimensions of `bw` are the same as those of `rngres`.

Tips

- This function assumes two-way propagation. For one-way propagation, you can find the required bandwidth by multiplying the output of this function by 2.

Algorithms

The function computes `c/(2*rngres)`.

Version History

Introduced in R2012b

References

[1] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

Functions

`bw2range` | `time2range` | `range2time`

Objects

`phased.FMCWWaveform`

Topics

“Automotive Adaptive Cruise Control Using FMCW Technology” (Radar Toolbox)

range2time

Convert propagation distance to propagation time

Syntax

```
t = range2time(r)
t = range2time(r,c)
```

Description

`t = range2time(r)` returns the time a signal takes to propagate a given distance. The propagation is assumed to be two-way, as in a monostatic radar system.

`t = range2time(r,c)` specifies the signal propagation speed.

Examples

PRF for Specified Unambiguous Range

Calculate the required PRF in Hertz for a monostatic radar system so that it can have a maximum unambiguous range of 15 km.

```
r = 15.0e3;
prf = 1/range2time(r)
```

```
prf = 9.9931e+03
```

Input Arguments

r — Signal range
array of nonnegative numbers

Signal range in meters, specified as an array of nonnegative numbers.

Data Types: `double`

c — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar in meters per second.

Data Types: `double`

Output Arguments

t — Propagation time
array of nonnegative numbers

Propagation time in seconds, returned as an array of nonnegative numbers. The dimensions of `t` are the same as those of `r`.

Algorithms

The function computes `2*r/c`.

Version History

Introduced in R2012b

References

[1] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`time2range` | `range2bw` | `phased.FMCWWaveform`

Topics

"Automotive Adaptive Cruise Control Using FMCW Technology" (Radar Toolbox)

range2tl

Compute underwater sound transmission loss from range

Syntax

`tl = range2tl(rng,freq,depth)`

Description

`tl = range2tl(rng,freq,depth)` returns the transmission loss, `tl`, for a sound wave of frequency `freq` arriving from a source at distance `rng`. The channel depth is `depth`. The transmission loss is due to geometrical spreading and frequency-dependent absorption. This function is the inverse of `tl2range`.

Examples

Estimate Transmission Loss from Range

Find the transmission loss (in dB) for a sonar operating at 2 kHz in a channel that is 200 m deep. The sound path is 1000.0 m long.

```
rng = 1000.0;
freq = 2000.0;
depth = 200;
tl = range2tl(rng,freq,depth)
```

```
tl = 50.1261
```

Input Arguments

`rng` — Distance from sound source to receiver
positive scalar

Distance from sound source to receiver, specified as a positive scalar. Units are in meters.

Example: `10e3`

Data Types: `double`

`freq` — Frequency of sound
positive scalar

Frequency of sound, specified as a positive scalar. Units are in Hz.

Example: `1e3`

Data Types: `double`

`depth` — Channel depth
positive scalar

Channel depth, specified as a positive scalar. Units are in meters.

Example: `200`

Data Types: `double`

Output Arguments

`tl` — Transmission loss
positive scalar

Transmission loss, returned as a positive scalar. Units are in dB.

Data Types: `double`

Limitations

- The transmission loss model assumes that seawater salinity is 35 ppt, pH is 8, and temperature is 10°C.
- The transmission loss model is valid for frequencies less than or equal to 2.0 MHz.

Version History

Introduced in R2017b

References

[1] Ainslie M. A. and J.G. McColm. "A simplified formula for viscous and chemical absorption in sea water." *Journal of the Acoustical Society of America*, Vol. 103, Number 3, 1998, pp. 1671--1672.

[2] Urick, Robert J. *Principles of Underwater Sound*, 3rd ed. Los Altos, CA:Peninsula Publishing, 1983.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`sonareqtl` | `sonareqsnr` | `sonareqsl` | `tl2range`

Topics
"Transmission Loss (TL)"
"Sonar Equation"

External Websites
http://resource.npl.co.uk/acoustics/techguides/seaabsorption/#content

rangeMigrationFMCW

Range migration image formation algorithm for frequency-modulated CW waveform

Syntax

```
slcimg = rangeMigrationFMCW(raw,waveform,fc,v,rc)
slcimg = rangeMigrationFMCW(raw,waveform,fc,v,rc,Name=Value)
```

Description

`slcimg = rangeMigrationFMCW(raw,waveform,fc,v,rc)` returns a single-look complex image of raw synthetic aperture radar (SAR) data obtained with a frequency modulated continuous wave (FMCW) waveform. The function uses the range migration algorithm.

`slcimg = rangeMigrationFMCW(raw,waveform,fc,v,rc,Name=Value)` specifies additional inputs using name-value arguments. You can specify the squint angle of the antenna and the signal propagation speed.

Examples

Single-Look Complex Image from Unfocused FMCW Data

Load a file that contains simulated unfocused synthetic aperture radar (SAR) data obtained with a frequency modulated continuous wave waveform sampled at 15 MHz. The waveform has a sweep time of 25 microseconds, a sweep bandwidth of 20 MHz, and sweeps in the `"Down"` direction.

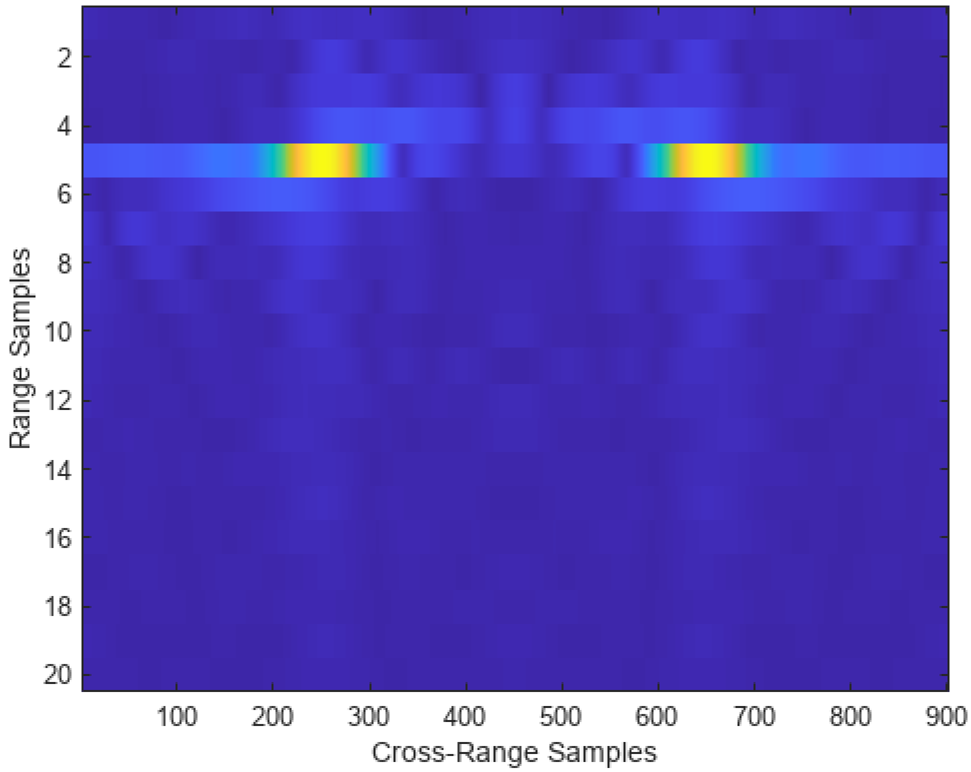
```
data = load("RangeMigrationFMCWExampleData");
raw = data.raw;

wvf = phased.FMCWWaveform(SampleRate=15e6, ...
    SweepTime=2.5e-5,SweepBandwidth=20e6,SweepDirection="Down");
```

The system works at an operating frequency of 3 GHz. The platform moves at a velocity of 200 meters per second and the distance between the beam center on the ground and the radar is 46.672 meters.

```
fc = 3e9;
v = 200;
rc = 46.672;
```

Generate a single-look complex image from the data. Plot the image and zoom in on the first 20 along-range samples.

```
slcimg = rangeMigrationFMCW(raw,wvf,fc,v,rc);

imagesc(abs(slcimg(1:20,:)))
xlabel("Cross-Range Samples")
ylabel("Range Samples")
```



Input Arguments

`raw` — Raw SAR data
matrix

Raw SAR data, specified as a matrix. The data is the unfocused in-phase and quadrature (I/Q) raw data collected by the SAR system. The rows of `raw` correspond to the along-range samples. The columns of `raw` correspond to the pulses received as the platform moves along the cross-range direction. `raw` must have at least two rows.

Data Types: `double`

`waveform` — Input waveform
`phased.FMCWWaveform` object

Input waveform, specified as a `phased.FMCWWaveform` object.

Example:
`phased.FMCWWaveform(SweepBandwidth=100e3,OutputFormat="Sweeps",NumSweeps=2)`

Data Types: `double`

`v` — Platform velocity
positive scalar

Platform velocity, specified as a positive scalar. Units are in meters per second.

Data Types: `double`

`fc` — Operating frequency
positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `2.8e9` Hz specifies a typical S-band operating-frequency value for airport and weather radar systems.

Data Types: `double`

`rc` — Distance between radar and beam center
positive scalar

Distance between radar and beam center on the ground, specified as a positive scalar. Units are in meters.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `SquintAngle=45,PropagationSpeed=343`

`SquintAngle` — Squint angle
`0` (default) | scalar in the range `(-90, 90)`

Squint angle of the antenna from the broadside direction in degrees, specified as a scalar in the range `(-90, 90)`.

Data Types: `double`

`PropagationSpeed` — Signal propagation speed
`physconst("LightSpeed")` (default) | positive scalar

Signal propagation speed in meters per second, specified as a positive scalar.

Example: `343` meters per second approximates the speed of sound at sea level and at a temperature of 20 °C under normal atmospheric conditions.

Data Types: `double`

Output Arguments

`slcimg` — Single-look complex image
matrix

Single-look complex (SLC) image, returned as a matrix. `slcimg` is the same size as `raw` and contains the focused data processed by the range migration algorithm.

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.FMCWWaveform` | `rangeMigrationLFM` | `rangeMigrationSFM`

Topics

"Stripmap Synthetic Aperture Radar (SAR) Image Formation" (Radar Toolbox)
"Squinted Spotlight Synthetic Aperture Radar (SAR) Image Formation" (Radar Toolbox)
"Synthetic Aperture Radar System Simulation and Image Formation" (Radar Toolbox)

rangeMigrationLFM

Range migration image formation algorithm for linear FM waveform

Syntax

```
slcimg = rangeMigrationLFM(raw,waveform,fc,v,rc)
slcimg = rangeMigrationLFM(raw,waveform,fc,v,rc,Name=Value)
```

Description

`slcimg = rangeMigrationLFM(raw,waveform,fc,v,rc)` returns a single-look complex image of raw synthetic aperture radar (SAR) data obtained with a linear frequency modulated (LFM) pulse waveform. The function uses the range migration algorithm.

`slcimg = rangeMigrationLFM(raw,waveform,fc,v,rc,Name=Value)` specifies additional inputs using name-value arguments. You can specify the squint angle of the antenna and the signal propagation speed.

Examples

Single-Look Complex Image from Unfocused LFM SAR Data

Load a file that contains simulated unfocused synthetic aperture radar (SAR) data obtained with a linear frequency modulated waveform. The waveform has a sample rate of 45 MHz, a pulse width of 3 microseconds, a pulse repetition frequency of 960 seconds, and a sweep bandwidth of c/10 Hz, where c is the numerical value of the speed of light.

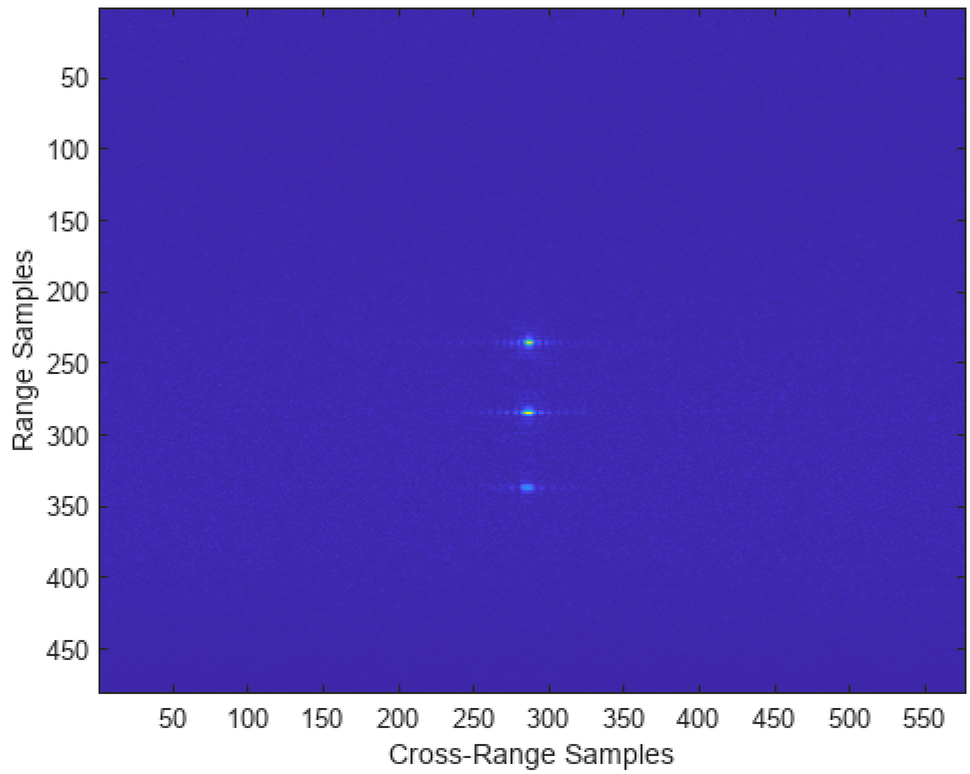
```
data = load("RangeMigrationLFMExampleData");
raw = data.raw;

wvf = phased.LinearFMWaveform(SampleRate=45e6, ...
    PulseWidth=3e-6,PRF=960,SweepBandwidth=physconst("LightSpeed")/10);
```

The system works at an operating frequency of 4 GHz. The platform moves at a velocity of 100 meters per second and the distance between the beam center on the ground and the radar is 1118 meters.

```
fc = 4e9;
v = 100;
rc = 1118;
```

Generate a single-look complex image from the data. Plot the image.

```
slcimg = rangeMigrationLFM(raw,wvf,fc,v,rc);

imagesc(abs(slcimg))
xlabel("Cross-Range Samples")
ylabel("Range Samples")
```



Input Arguments

`raw` — Raw SAR data
matrix

Raw SAR data, specified as a matrix. The data is the unfocused in-phase and quadrature (I/Q) raw data collected by the SAR system. The rows of `raw` correspond to the along-range samples. The columns of `raw` correspond to the pulses received as the platform moves along the cross-range direction. `raw` must have at least two rows.

Data Types: `double`

`waveform` — Input waveform
`phased.LinearFMWaveform` object

Input waveform, specified as a `phased.LinearFMWaveform` object.

Example:
`phased.LinearFMWaveform(SampleRate=500e3,'SweepBandwidth',200e3,PulseWidth=1e-3,PRF=1e3)`

Data Types: `double`

`fc` — Operating frequency
positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `2.8e9` Hz specifies a typical S-band operating-frequency value for airport and weather radar systems.

Data Types: `double`

`v` — Platform velocity
positive scalar

Platform velocity, specified as a positive scalar. Units are in meters per second.

Data Types: `double`

`rc` — Distance between radar and beam center
positive scalar

Distance between radar and beam center on the ground, specified as a positive scalar. Units are in meters.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `SquintAngle=45,PropagationSpeed=343`

`SquintAngle` — Squint angle
`0` (default) | scalar in the range `(-90, 90)`

Squint angle of the antenna from the broadside direction in degrees, specified as a scalar in the range `(-90, 90)`.

Data Types: `double`

`PropagationSpeed` — Signal propagation speed
`physconst("LightSpeed")` (default) | positive scalar

Signal propagation speed in meters per second, specified as a positive scalar.

Example: `343` meters per second approximates the speed of sound at sea level and at a temperature of 20 °C under normal atmospheric conditions.

Data Types: `double`

Output Arguments

`slcimg` — Single-look complex image
matrix

Single-look complex (SLC) image, returned as a matrix. `slcimg` is the same size as `raw` and contains the focused data processed by the range migration algorithm.

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.LinearFMWaveform` | `rangeMigrationFMCW` | `rangeMigrationSFM`

Topics

"Stripmap Synthetic Aperture Radar (SAR) Image Formation" (Radar Toolbox)
"Squinted Spotlight Synthetic Aperture Radar (SAR) Image Formation" (Radar Toolbox)
"Synthetic Aperture Radar System Simulation and Image Formation" (Radar Toolbox)

rangeMigrationSFM

Range migration image formation algorithm for stepped FM waveform

Syntax

```
slcimg = rangeMigrationSFM(raw,waveform,fc,v,rc)
slcimg = rangeMigrationSFM(raw,waveform,fc,v,rc,Name=Value)
```

Description

`slcimg = rangeMigrationSFM(raw,waveform,fc,v,rc)` returns a single-look complex image of raw synthetic aperture radar (SAR) data obtained with a stepped frequency modulated (SFM) pulse waveform. The function uses the range migration algorithm.

`slcimg = rangeMigrationSFM(raw,waveform,fc,v,rc,Name=Value)` specifies additional inputs using name-value arguments. You can specify the squint angle of the antenna and the signal propagation speed.

Examples

Single-Look Complex Image from Unfocused SFM SAR Data

Load a file that contains simulated unfocused synthetic aperture radar (SAR) data obtained with a stepped frequency modulated waveform. The waveform has a sample rate of 200 MHz, a pulse width of 250 nanoseconds, a pulse repetition frequency of 2/3 MHz, a frequency step size of 200 kHz, and 128 frequency steps.

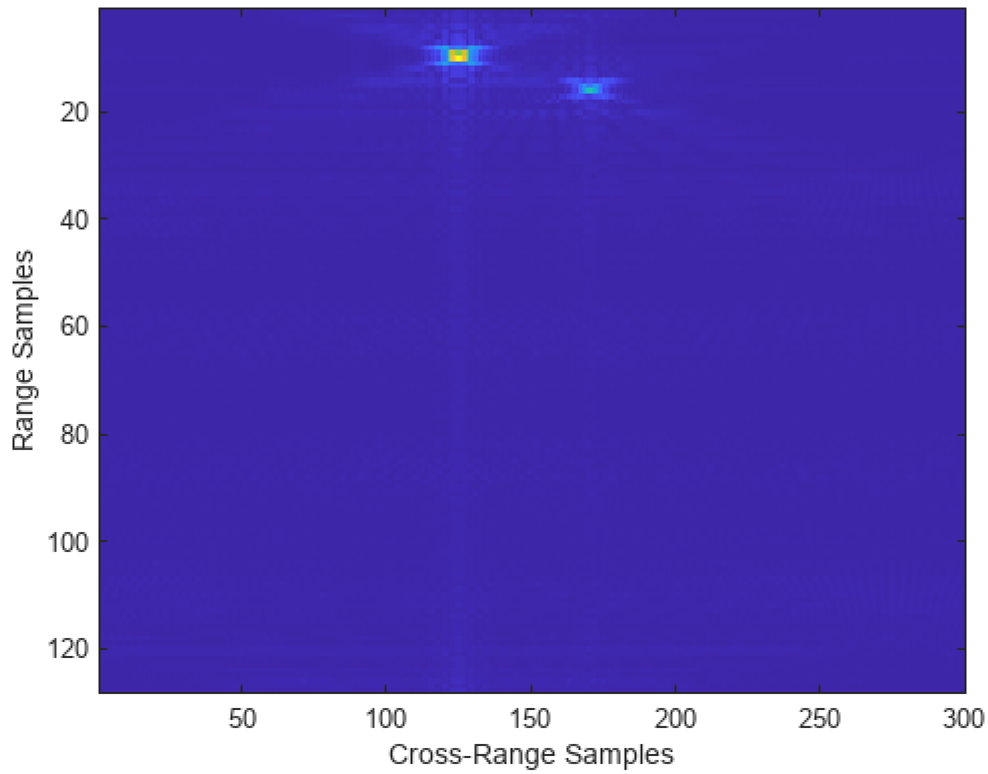
```
data = load('RangeMigrationSFMExampleData.mat');
raw = data.raw;

wvf = phased.SteppedFMWaveform(SampleRate=200e6, ...
    PulseWidth=250e-9,PRF=2e6/3,FrequencyStep=200e3,NumSteps=128);
```

The system works at an operating frequency of 4 GHz. The platform moves at a velocity of 400 meters per second and the distance between the beam center on the ground and the radar is 60 meters.

```
fc = 4e9;
v = 400;
rc = 60;
```

Generate a single-look complex image from the data. Plot the image.

```
slcimg = rangeMigrationSFM(raw,wvf,fc,v,rc);

imagesc(abs(slcimg))
xlabel('Cross-Range Samples')
ylabel('Range Samples')
```



Input Arguments

`raw` — Raw SAR data
matrix

Raw SAR data, specified as a matrix. The data is the unfocused in-phase and quadrature (I/Q) raw data collected by the SAR system. The rows of `raw` correspond to the frequency steps. The columns of `raw` correspond to the bursts. `raw` must have at least two rows.

Data Types: `double`

`waveform` — Input waveform
`phased.SteppedFMWaveform` object

Input waveform, specified as a `phased.SteppedFMWaveform` object.

Example: `phased.SteppedFMWaveform(NumSteps=3,FrequencyStep=20e3)`

Data Types: `double`

`v` — Platform velocity
positive scalar

Platform velocity, specified as a positive scalar. Units are in meters per second.

Data Types: `double`

fc — Operating frequency
positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: 2.8e9 Hz specifies a typical S-band operating-frequency value for airport and weather radar systems.

Data Types: double

rc — Distance between radar and beam center
positive scalar

Distance between radar and beam center on the ground, specified as a positive scalar. Units are in meters.

Data Types: double

Name-Value Pair Arguments

Specify optional pairs of arguments as Name1=Value1,...,NameN=ValueN, where Name is the argument name and Value is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: SquintAngle=45,PropagationSpeed=343

SquintAngle — Squint angle
0 (default) | scalar in the range (-90, 90)

Squint angle of the antenna from the broadside direction in degrees, specified as a scalar in the range (-90, 90).

Data Types: double

PropagationSpeed — Signal propagation speed
physconst("LightSpeed") (default) | positive scalar

Signal propagation speed in meters per second, specified as a positive scalar.

Example: 343 meters per second approximates the speed of sound at sea level and at a temperature of 20 °C under normal atmospheric conditions.

Data Types: double

Output Arguments

slcimg — Single-look complex image
matrix

Single-look complex (SLC) image, returned as a matrix. slcimg is the same size as raw and contains the focused data processed by the range migration algorithm.

Version History

Introduced in R2022a

References

[1] Cumming, Ian G., and Frank Hay-chee Wong. *Digital Processing of Synthetic Aperture Radar Data: Algorithms and Implementation*. Artech House Remote Sensing Library. Boston: Artech House, 2005.

[2] Tolman, Matthew A. "A Detailed Look at the Omega-k Algorithm for Processing Synthetic Aperture Radar Data.". MS thesis, Brigham Young University, 2008.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.SteppedFMWaveform` | `rangeMigrationFMCW` | `rangeMigrationLFM`

Topics

“Stripmap Synthetic Aperture Radar (SAR) Image Formation” (Radar Toolbox)
“Squinted Spotlight Synthetic Aperture Radar (SAR) Image Formation” (Radar Toolbox)
“Synthetic Aperture Radar System Simulation and Image Formation” (Radar Toolbox)

rangeDopplerImagerLFM

Range migration image formation algorithm for linear FM waveform

Syntax

```
slcimg = rangeDopplerImagerLFM(raw,waveform,fc,v,rc)
slcimg = rangeDopplerImagerLFM(raw,waveform,fc,v,rc,Name=Value)
```

Description

`slcimg = rangeDopplerImagerLFM(raw,waveform,fc,v,rc)` returns a single-look complex image `slcimg` generated from raw synthetic aperture radar (SAR) data obtained using a linear frequency modulated (LFM) pulse `phased.LinearFMWaveform` `waveform`. `fc` specifies the operating frequency. `v` specifies the platform speed. `rc` is the radar-to-beamcenter distance. The function uses the range migration algorithm.

`slcimg = rangeDopplerImagerLFM(raw,waveform,fc,v,rc,Name=Value)` specifies additional inputs using name-value arguments. Name-value pairs let you specify squint angle, `sqa`, and propagation speed, `c`.

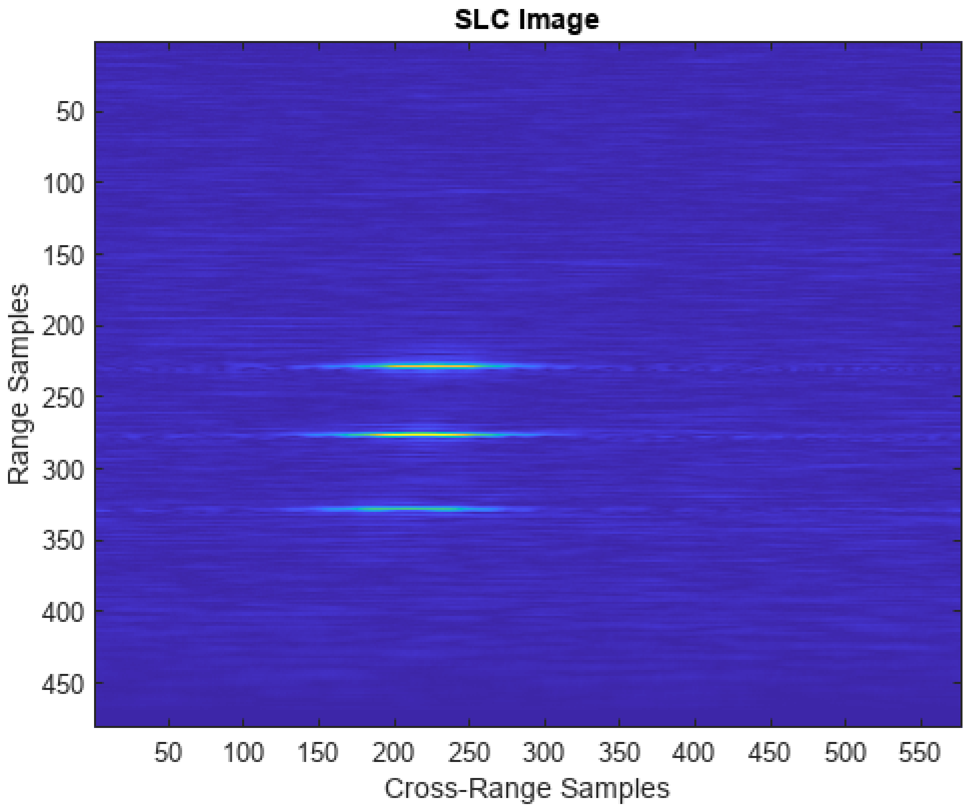
Examples

Generate Image for Unfocused SAR

Generate a complex-valued single-look SAR image from simulated unfocused raw data. The radar transmits an LFM waveform. The LFM waveform has a 3 microsec duration with a PRF of 960 Hz. The radar has an operating frequency of 9 GHZ with a pulse sweep bandwidth of 29.979 MHz. The radar platform moves at 100 m/s. The radar-to-beamcenter distance is 1.118 km.

FIrst, load the raw image data from a mat-file. The raw data consists of a 481-by-577 complex-valued matrix. Then compute the focused SAR image.

```
load('RangeMigrationLFMExampleData.mat')
slcimg = rangeDopplerImagerLFM(raw,waveform,fc,v,rc);
imagesc(abs(slcimg))
title('SLC Image')
xlabel('Cross-Range Samples')
ylabel('Range Samples')
```



Input Arguments

raw — Unfocused IQ raw SAR data
M-by-*N* complex-valued matrix

Unfocused IQ raw SAR data, specified as an *M*-by-*N* complex-valued matrix. The data is the unfocused in-phase and quadrature (I/Q) raw data collected by the SAR system. Rows of `raw` correspond to along-range samples. The columns of `raw` correspond to the pulses received as the platform moves along the cross-range direction. `raw` must have at least two rows.

Data Types: `double`

waveform — Input waveform
`phased.LinearFMWaveform` object

Input waveform, specified as a `phased.LinearFMWaveform` object.

Example:
`phased.LinearFMWaveform(SampleRate=500e3,'SweepBandwidth',200e3,PulseWidth=1e-3,PRF=1e3)`

Data Types: `double`

fc — Operating frequency
positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Example: `2.8e9` Hz specifies a typical S-band operating-frequency value for airport and weather radar systems.

Data Types: `double`

v — Platform velocity
positive scalar

Platform velocity, specified as a positive scalar. Units are in meters per second.

Data Types: `double`

`rc` — Distance between radar and beam center
positive scalar

Distance between radar and beam center on the ground, specified as a positive scalar. Units are in meters.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `SquintAngle=45,PropagationSpeed=343`

`SquintAngle` — Squint angle
`0` (default) | scalar in the range `(-90, 90)`

Squint angle of the antenna from the broadside direction in degrees, specified as a scalar in the range `(-90, 90)`.

Data Types: `double`

`PropagationSpeed` — Signal propagation speed
`physconst("LightSpeed")` (default) | positive scalar

Signal propagation speed in meters per second, specified as a positive scalar.

Example: `343` meters per second approximates the speed of sound at sea level and at a temperature of 20 °C under normal atmospheric conditions.

Data Types: `double`

Output Arguments

`slcimg` — Single-look complex image
matrix

Single-look complex (SLC) image, returned as a matrix. `slcimg` is the same size as `raw` and contains the focused data processed by the range migration algorithm.

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`phased.LinearFMWaveform` | `rangeMigrationFMCW` | `rangeMigrationSFM`

Topics

"Stripmap Synthetic Aperture Radar (SAR) Image Formation" (Radar Toolbox)
"Squinted Spotlight Synthetic Aperture Radar (SAR) Image Formation" (Radar Toolbox)
"Synthetic Aperture Radar System Simulation and Image Formation" (Radar Toolbox)

rotpat

Rotate radiation pattern

Syntax

```
rpat = rotpat(pat,az,el,rotax)
rpat = rotpat(pat,az,el,rotax,expval)
```

Description

`rpat = rotpat(pat,az,el,rotax)` rotates a radiation pattern, `pat`, into a new pattern, `rpat`, whose boresight is aligned with the *x*-axis of a new local coordinate system defined by `rotax`. `az` and `el` specify the azimuth and elevation angles at which the original pattern is sampled.

`rpat = rotpat(pat,az,el,rotax,expval)` also specifies an extrapolated value to be used when `az` and `el` do not cover the entire 3-D space.

Tip You can use this function to rotate real and complex scalar radiation patterns as well as the orthogonal components of polarized fields. To rotate polarized fields, rotate the horizontal and vertical polarization components separately.

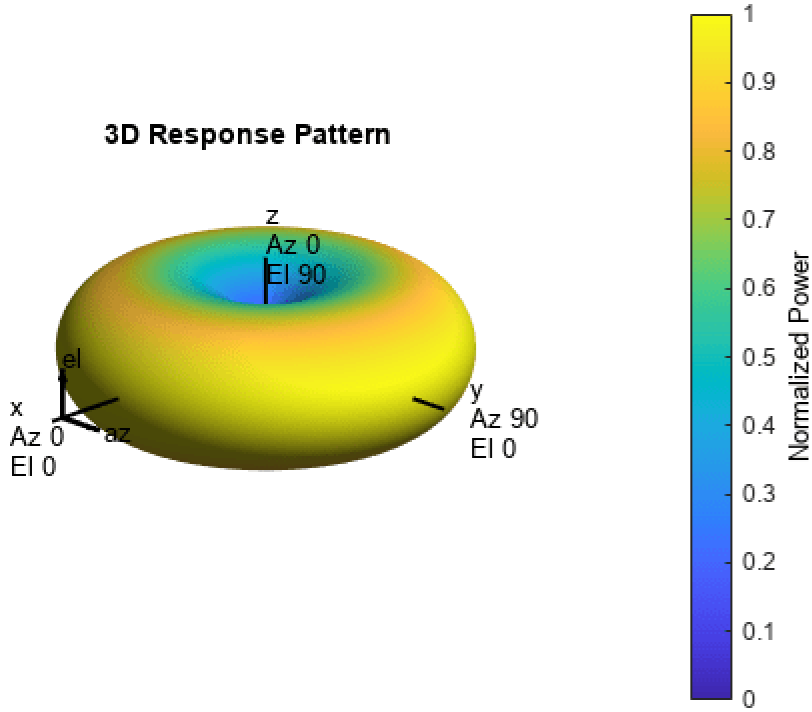
Examples

Rotate Pattern of Short-Dipole Antenna

Use a short-dipole antenna to create a polarized radiation pattern. Rotate the pattern and use the rotated pattern as the radiation pattern of a custom antenna.

Create a `phased.ShortDipoleAntennaElement` antenna object with default properties. The short-dipole antenna radiates polarized radiation. Obtain and display the radiation for all directions.

```
antenna1 = phased.ShortDipoleAntennaElement;
el = -90:90;
az = -180:180;
pat_h = zeros(numel(el),numel(az),'like',1+1i);
pat_v = pat_h;
fc = 3e8;
for m = 1:numel(el)
    temp = antenna1(fc,[az;el(m)*ones(1,numel(az))]);
    pat_h(m,:) = temp.H;
    pat_v(m,:) = temp.V;
end
pattern(antenna1,fc,'Type','Power')
```

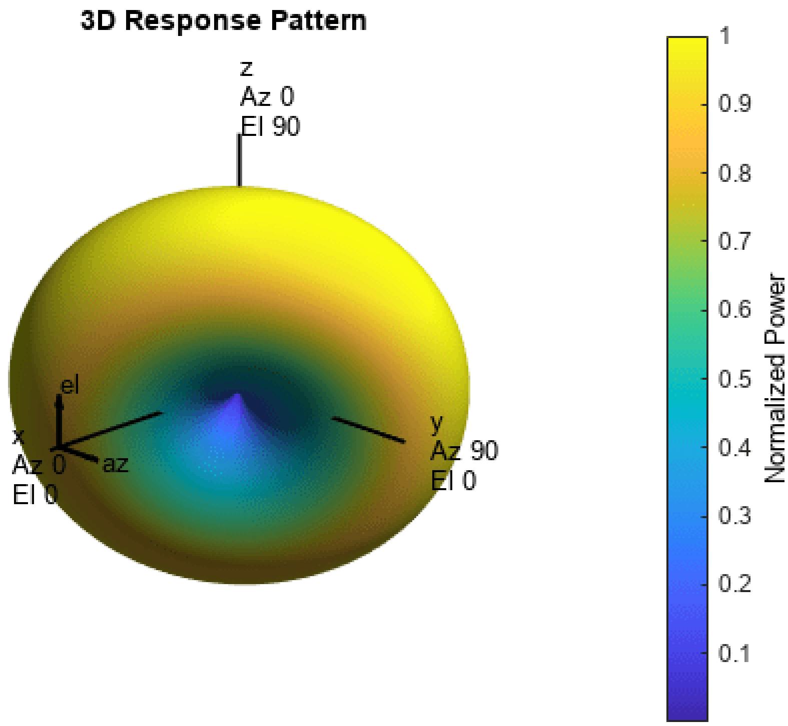


Rotate the antenna pattern around the *y*-axis by 135 degrees followed by a rotation around the *x*-axis by 65 degrees.

```
newax = rotx(65)*roty(135);
pat2_h = rotpat(pat_h,az,el,newax);
pat2_v = rotpat(pat_v,az,el,newax);
```

Insert the rotated pattern into a `phased.CustomAntennaElement` object. Set the antenna polarization properties so that the element radiates horizontal and vertical polarized fields. Then display the rotated pattern in three dimensions.

```
antenna2 = phased.CustomAntennaElement( ...
    'SpecifyPolarizationPattern',true, ...
    'HorizontalMagnitudePattern',mag2db(abs(pat2_h)), ...
    'HorizontalPhasePattern',rad2deg(angle(pat2_h)), ...
    'VerticalMagnitudePattern',mag2db(abs(pat2_v)), ...
    'VerticalPhasePattern',rad2deg(angle(pat2_v)));
pattern(antenna2,fc,'Type','Power')
```



Rotate Pattern of Cosine Antenna

Create a radiation pattern for a cosine antenna using a `phased.CosineAntennaElement` object. Rotate the pattern to use in a `phased.CustomAntennaElement` antenna object.

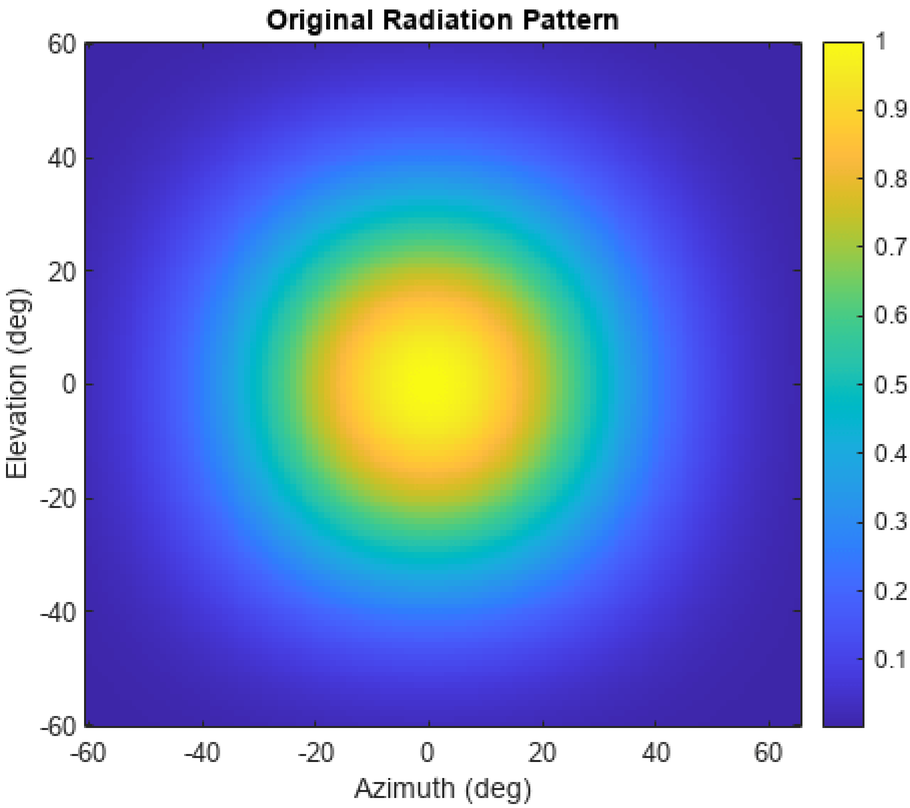
First obtain the radiation pattern for a `phased.CosineAntennaElement` object over a limited range of directions. The field is not polarized.

```
antenna1 = phased.CosineAntennaElement('CosinePower',[5,5]);
az = -60:65;
el = -60:60;
pat = zeros(numel(el),numel(az),'like',1);
fc = 300e6;
for m = 1:numel(el)
    temp = antenna1(fc,[az;el(m)*ones(1,numel(az))]);
    pat(m,:) = temp;
end
```

Display the original pattern.

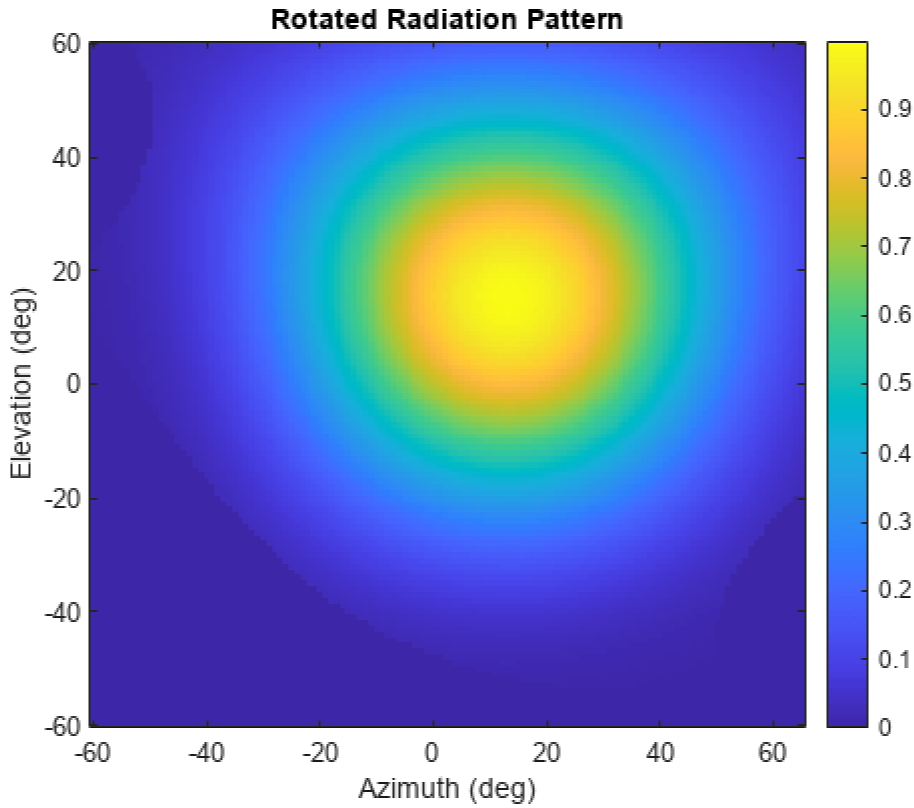
```
imagesc(az,el,abs(pat))
axis xy
axis equal
axis tight
xlabel('Azimuth (deg)')
```

```
ylabel('Elevation (deg)')
title('Original Radiation Pattern')
colorbar
```



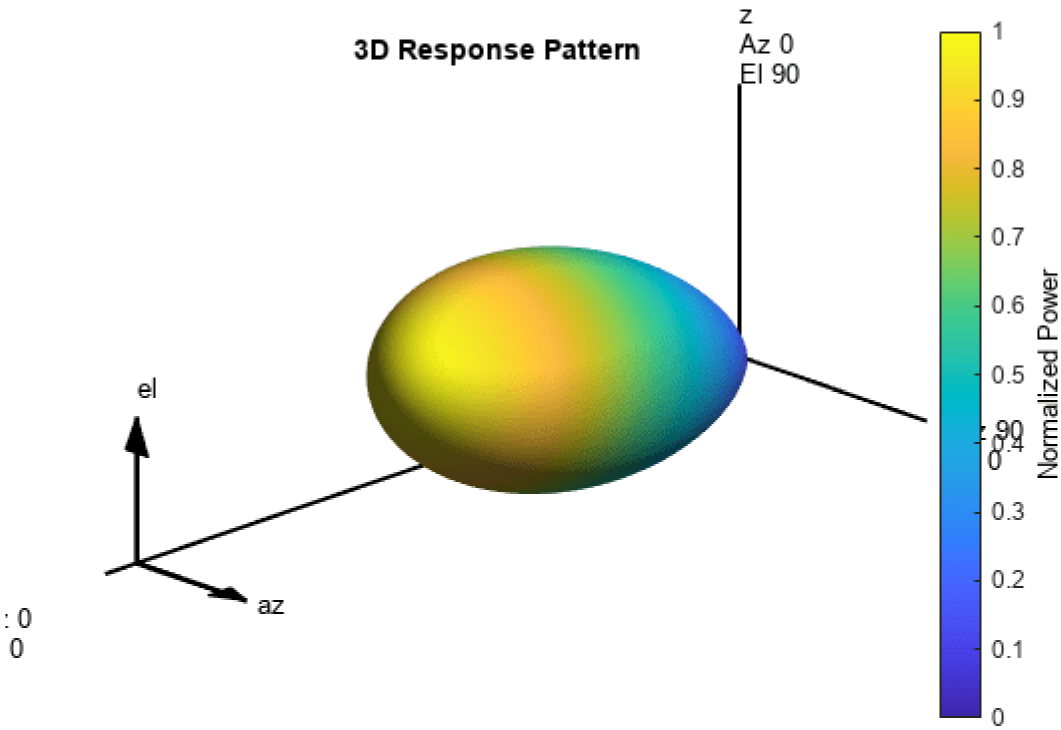
Rotate the antenna pattern by 20 degrees around the *z*-axis and 50 degrees around the *x*-axis. Then display the rotated pattern.

```
newax = rotx(50)*rotz(20);
rpat = rotpat(pat,az,el,newax);
imagesc(az,el,abs(rpat))
axis xy
axis equal
axis tight
xlabel('Azimuth (deg)')
ylabel('Elevation (deg)')
title('Rotated Radiation Pattern')
colorbar
```



Use the rotated pattern in a custom antenna element and display the pattern in 3-D.

```
antenna2 = phased.CustomAntennaElement( ...
    'AzimuthAngles',az,'ElevationAngles',el,'SpecifyPolarizationPattern',false, ...
    'MagnitudePattern',mag2db(abs(rpat)), ...
    'PhasePattern',zeros(size(rpat)));
pattern(antenna2,fc,'Type','Power')
```



Input Arguments

pat — Radiation pattern

complex-valued *N*-by-*M* matrix | complex-valued *N*-by-*M*-by-*L* array

Radiation pattern, specified as a complex-valued *N*-by-*M* matrix or complex-valued *N*-by-*M*-by-*L* array. *N* is the length of the `el` vector and *M* is the length of the `az` vector. Each column corresponds to one of the azimuth angles specified in the `az` argument. Each row corresponds to one of the elevation angles specified in the `el` argument. You can specify multiple radiation patterns using *L* pages. For example, you can use pages to specify radiation patterns at different frequencies. The main lobe of each pattern is assumed to point along the *x*-axis. Units are in meters-squared.

Data Types: `double`

az — Azimuth angles

`-180:180` (default) | 1-by-*M* real-valued row vector

Azimuth angles for computing 3-D radiation pattern, specified as a 1-by-*M* real-valued row vector where *M* is the number of azimuth angles. Each entry corresponds to one of the columns of the matrix specified in the `pat` argument. Angle units are in degrees. Azimuth angles must lie between –180° and 180°, inclusive.

The azimuth angle is the angle between the *x*-axis and the projection of the direction vector onto the *xy*-plane. The azimuth angle is positive when measured from the *x*-axis toward the *y*-axis.

Example: `-45:2:45`

Data Types: `double`

`el` — Elevation angles
`-90:90` (default) | 1-by-*N* real-valued row vector

Elevation angles for computing directivity and pattern, specified as a 1-by-*N* real-valued row vector where *N* is the number of elevation angles. Each entry corresponds to one of the rows of the matrix specified in the `pat` argument. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive.

The elevation angle is the angle between the direction vector and *xy*-plane. The elevation angle is positive when measured toward the *z*-axis.

Example: `-75:1:70`

Data Types: `double`

`rotax` — Rotation matrix
real-valued orthonormal 3-by-3 matrix | real-valued orthonormal 3-by-3-by-*P* array of orthonormal matrices

Rotation matrix, specified as a real-valued orthonormal 3-by-3 matrix or a real-valued 3-by-3-by-*P* array. The columns represent the *x*, *y*, and *z* directions of the rotated coordinate system with respect to the original coordinate system. The *P* pages specify different rotation matrices.

This table describes how dimensions of the output pattern `rpat` depend on the dimensions of the `pat` and `rotax` arguments.

Dimensions of `rpat`

Dimensions of pat	Dimensions of rotax	
	3-by-*3*	*3*-by-*3*-by-*P*
M-by-*N*	Rotate a single pattern by a single rotation matrix. Output dimensions of `rpat` are *M*-by-*N*.	Rotate a single pattern by *P* different rotation matrices. Output dimensions of `rpat` are *M*-by-*N*-by-*P*.
M-by-*N*-by-*L*	Rotate *L* patterns by the same rotation matrix. Output dimensions of `rpat` are *M*-by-*N*-by-*L*.	In this case, *P* must equal *L* and the function rotates each pattern by the corresponding rotation matrix. Output dimensions of `rpat` are *M*-by-*N*-by-*L*.

Example: `rotx(45)*roty(30)`

Data Types: `double`

`expval` — Extrapolation value
`0` (default) | scalar

Extrapolation value, specified as a scalar. This scalar is the extrapolated value when the rotated patterns do not fill the entire 3-D space specified by `az` and `el`. In general, consider setting `expval` to 0 if the pattern is specified in a linear scale or `-inf` if the pattern is specified in a dB scale.

Example: `-inf`

Data Types: `double`

Output Arguments

`rpat` — Rotated radiation pattern
complex-valued *N*-by-*M* matrix | complex-valued *N*-by-*M*-by-*P* array

Rotated radiation pattern, returned as a complex-valued *N*-by-*M* matrix or complex-valued *N*-by-*M*-by-*P* array. *N* is the length of the `el` vector. *M* is the length of the `az` vector. The dimensionality of `pat` and `rotax` determine the value of *P* as discussed in the `rotax` input argument. Units are in meters-squared.

Data Types: `double`

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`rotx` | `roty` | `rotz`

tl2range

Compute range from underwater transmission loss

Syntax

```
rng = tl2range(tl,freq,depth)
```

Description

`rng = tl2range(tl,freq,depth)` returns the range, `rng`, to the source of a sound wave with frequency `freq` from the transmission loss, `tl`. The channel depth is `depth` and the sound frequency is `freq`. The transmission loss is due to geometrical spreading and frequency-dependent absorption. This function is the inverse of `range2tl` function.

Examples

Estimate Range from Transmission Loss

Find the distance traveled by a sound wave with a transmission loss of 50 dB. The sonar operates at 2 kHz in a channel 200 m deep.

```
tl = 50.0;
freq = 2000.0;
depth = 200.0;
rng = tl2range(tl,freq,depth)
```

```
rng = 972.1666
```

Input Arguments

`tl` — Transmission loss from source to receiver
positive scalar

Transmission loss from source to receiver, specified as a positive scalar. Units are in dB.

Data Types: `double`

`freq` — Frequency of sound
positive scalar less than or equal to 2 MHz

Frequency of sound, specified as a positive scalar less than or equal to 2 MHz. Units are in Hz.

Example: `1e3`

Data Types: `double`

`depth` — Depth of sound channel
positive scalar

Depth of sound channel, specified as a positive scalar. Units are in meters.

Example: `200`

Data Types: `double`

Output Arguments

`rng` — Distance from source to receiver
positive scalar

Distance from source to receiver, returned as a positive scalar. Units are in meters.

Data Types: `double`

Limitations

- The transmission loss model assumes that seawater salinity is 35 ppt, pH is 8, and temperature is 10°C.
- The transmission loss model is valid for frequencies less than or equal to 2.0 MHz.

Version History

Introduced in R2017b

References

[1] Ainslie M. A. and J.G. McColm. "A simplified formula for viscous and chemical absorption in sea water." *Journal of the Acoustical Society of America*, Vol. 103, Number 3, 1998, pp. 1671--1672.

[2] Urick, Robert J, *Principles of Underwater Sound*, 3rd ed. Peninsula Publishing, Los Altos, CA, 1983.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`sonareqtl` | `sonareqsnr` | `sonareqsl` | `range2tl`

Topics
"Transmission Loss (TL)"
"Sonar Equation"

External Websites
http://resource.npl.co.uk/acoustics/techguides/seaabsorption/#content

rangeangle

Range and angle calculation

Syntax

```
[rng,ang] = rangeangle(pos)
[rng,ang] = rangeangle(pos,refpos)
[rng,ang] = rangeangle(pos,refpos,refaxes)
[rng,ang] = rangeangle( ___ ,model)
```

Description

The function `rangeangle` determines the propagation path length and path direction of a signal from a source point or set of source points to a reference point. The function supports two propagation models – the *free space* model and the *two-ray* model. The *free space* model is a single line-of-sight path from a source point to a reference point. The *two-ray* multipath model generates two paths. The first path follows the free-space path. The second path is a reflected path off a boundary plane at $z = 0$. Path directions are defined with respect to either the global coordinate system at the reference point or a local coordinate system at the reference point. Distances and angles at the reference point do not depend upon which direction the signal is travelling along the path.

`[rng,ang] = rangeangle(pos)` returns the propagation path length, `rng`, and direction angles, `ang`, of a signal path from a source point or set of source points, `pos`, to the origin of the global coordinate system. The direction angles are the azimuth and elevation with respect to the global coordinate axes at the origin. Signals follow a line-of-sight path from the source point to the origin. The line-of-sight path corresponds to the geometric straight line between the points.

`[rng,ang] = rangeangle(pos,refpos)` also specifies a reference point or set of reference points, `refpos`. `rng` now contains the propagation path length from the source points to the reference points. The direction angles are the azimuth and elevation with respect to the global coordinate axes at the reference points. You can specify multiple points and multiple reference points.

`[rng,ang] = rangeangle(pos,refpos,refaxes)` also specifies local coordinate system axes, `refaxes`, at the reference points. Direction angles are the azimuth and elevation with respect to the local coordinate axes centered at `refpos`.

`[rng,ang] = rangeangle( ___ ,model)`, also specifies a propagation model. When `model` is set to `"freespace"`, the signal propagates along a line-of-sight path from source point to reception point. When `model` is set to `"two-ray"`, the signal propagates along two paths from source point to reception point. The first path is the line-of-sight path. The second path is the reflecting path. In this case, the function returns the distances and angles for two paths for each source point and corresponding reference point.

Examples

Range and Angle Computation

Compute the range and angle of a target located at *(1000,2000,50)* meters from the origin.

```
TargetLoc = [1000;2000;50];
[tgtrng,tgtang] = rangeangle(TargetLoc)
```

```
tgtrng = 2.2366e+03

tgtang = 2×1

   63.4349
    1.2810
```

Range and Angle With Respect to Local Origin

Compute the range and angle of a target located at *(1000,2000,50)* meters with respect to a local origin at *(100,100,10)* meters.

```
TargetLoc = [1000;2000;50];
Origin = [100;100;10];
[tgtrng,tgtang] = rangeangle(TargetLoc,Origin)
```

```
tgtrng = 2.1028e+03

tgtang = 2×1

   64.6538
    1.0900
```

Range and Angle With Respect to Local Coordinates

Compute the range and angle of a target located at *(1000,2000,50)* meters but with respect to a local coordinate system origin at *(100,100,10)* meters. Choose a local coordinate reference frame that is rotated about the z-axis by 45° from the global coordinate axes.

```
targetpos = [1000;2000;50];
origin = [100;100;10];
refaxes = [1/sqrt(2) -1/sqrt(2) 0; 1/sqrt(2) 1/sqrt(2) 0; 0 0 1];
[tgtrng,tgtang] = rangeangle(targetpos,origin,refaxes)
```

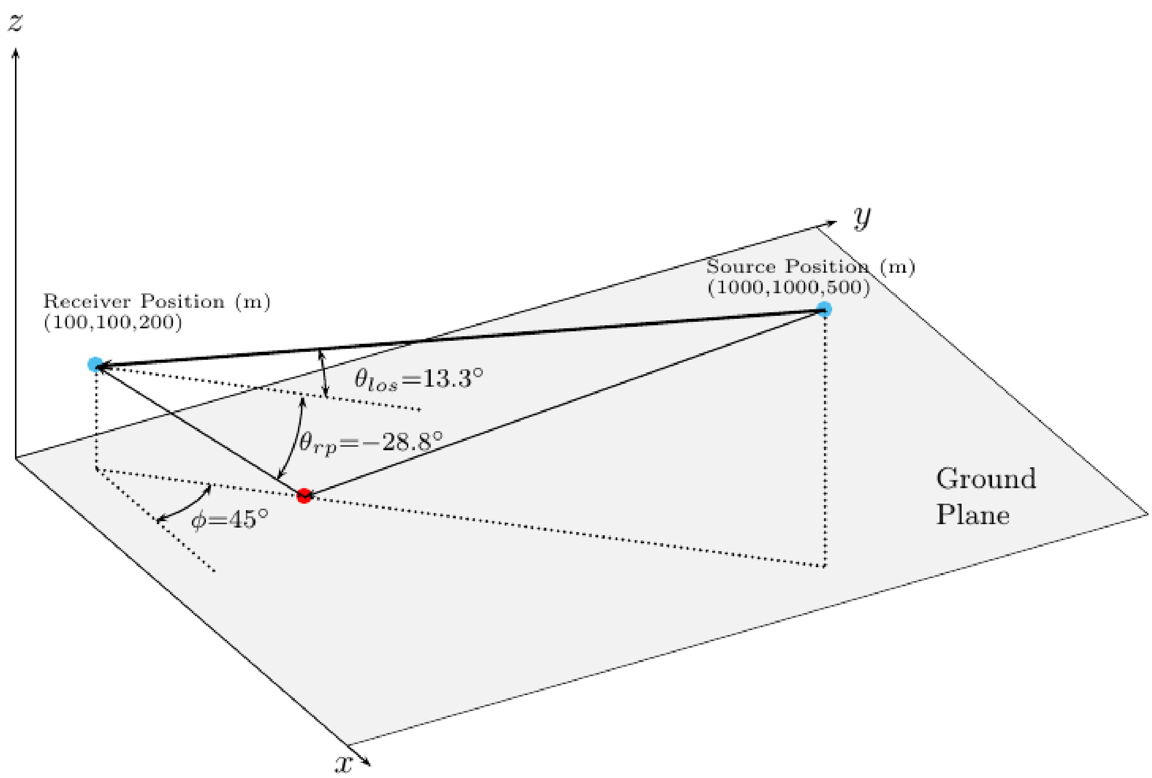
```
tgtrng = 2.1028e+03

tgtang = 2×1

   19.6538
    1.0900
```

Two-Ray Range and Angle

Compute the two-ray propagation distances and arrival angles of rays from a source located at (1000, 1000, 500) meters from the origin. The receiver is located at (100, 100, 200) meters from the origin.

```
sourceLoc = [1000;1000;500];
receiverLoc = [100;100;200];
[sourcerngs,sourceangs] = rangeangle(sourceLoc,receiverLoc,"two-ray")
```

```
sourcerngs = 1×2
10³ ×

    1.3077    1.4526


sourceangs = 2×2

   45.0000   45.0000
   13.2627  -28.8096
```



Find the range and angle of the same target with the same origin but with respect to a local coordinate axes. The local coordinate axes are rotated around the z-axis by 45 degrees from the global coordinate axes.

```
refaxes = rotz(45);
[sourcerngs,sourceangs] = rangeangle(sourceLoc,receiverLoc,refaxes,"two-ray")
```

```
sourcerngs = 1×2
10³ ×

    1.3077    1.4526


sourceangs = 2×2

         0         0
   13.2627  -28.8096
```

Range and Angle With Respect to Two Origins

Compute the ranges and angles of two targets located at (1000,200,500) and (2500,80,-100) meters with respect to two local origins at (100,300,-40) and (500,-60,10) meters. Specify two different sets of local axes.

```
targetPos = [1000,2500;200,80;500,-100];
origins = [100,500;300,-60;-40,10];
ax(:,:,1) = rotx(40)*rotz(10);
ax(:,:,2) = roty(5)*rotx(10);
[tgtrng,tgtang] = rangeangle(targetPos,origins,ax)
```

```
tgtrng = 1×2
10³ ×

    1.0543    2.0079


tgtang = 2×2

    6.7285    4.2597
   26.9567    1.1254
```

Input Arguments

pos — Source point position
real-valued 3-by-1 vector | real-valued 3-by-*N* matrix

Source point position in meters, specified as a real-valued 3-by-1 vector or a real-valued 3-by-*N* matrix. A matrix represents multiple source points. The columns contain the Cartesian coordinates of *N* points in the form `[x;y;z]`.

When `pos` is a 3-by-*N* matrix, you must specify `refpos` as a 3-by-*N* matrix for *N* reference positions. If all the reference points are identical, you can specify `refpos` by a single 3-by-1 vector.

Example: `[1000;2000;50]`

Data Types: `double`

`refpos` — Reference point position
`[0;0;0]` (default) | real-valued 3-by-1 vector | real-valued 3-by-*N* matrix

Reference point position in meters, specified as a real-valued 3-by-1 vector or a real-valued 3-by-*N* matrix. A matrix represents multiple reference points. The columns contain the Cartesian coordinates of *N* points ins the form `[x;y;z]`.

When `refpos` is a 3-by-*N* matrix, you must specify `pos` as a 3-by-*N* matrix for *N* source positions. If all the source points are identical, you can specify `pos` by a single 3-by-1 vector.

Position units are meters.

Example: `[100;100;10]`

Data Types: `double`

`refaxes` — Local coordinate system axes
`[1 0 0;0 1 0;0 0 1]` (default) | real-valued 3-by-3 matrix | real-valued 3-by-3-by-*N* array

Local coordinate system axes, specified as a real-valued 3-by-3 matrix or a 3-by-3-by-*N* array. For an array, each page corresponds to a local coordinate axes at each reference point. The columns in `refaxes` specify the direction of the coordinate axes for the local coordinate system in Cartesian coordinates. *N* must match the number of columns in `pos` or `refpos` when these dimensions are greater than one.

Example: `rotz(45)`

Data Types: `double`

`model` — Propagation model
`"freespace"` (default) | `"two-ray"`

Propagation model, specified as `"freespace"` or `"two-ray"`. Choosing `"freespace"` invokes the free space propagation model. Choosing `"two-ray"` invokes the two-ray propagation model.

Data Types: `char` | `string`

Output Arguments

`rng` — Propagation range
real-valued 1-by-*N* vector | real-valued 1-by-2*N* vector

Propagation range in meters, returned as a real-valued 1-by-*N* vector or real-valued 1-by-2*N* vector.

- When `model` is set to `"freespace"`, the size of `rng` is 1-by-*N*. The propagation range is the length of the direct path from the position defined in `pos` to the corresponding reference position defined in `refpos`.
- When `model` is set to `"two-ray"`, `rng` contains the ranges for the direct path and the reflected path. Alternate columns of `rng` refer to the line-of-sight path and reflected path, respectively for the same source-reference point pair.

`ang` — Azimuth and elevation angles
real-valued 2-by-*N* matrix | real-valued 2-by-2*N* matrix

Azimuth and elevation angles in degrees, returned as a 2-by-*N* matrix or 2-by-2*N* matrix. Each column represents a direction angle in the form `[azimuth;elevation]`.

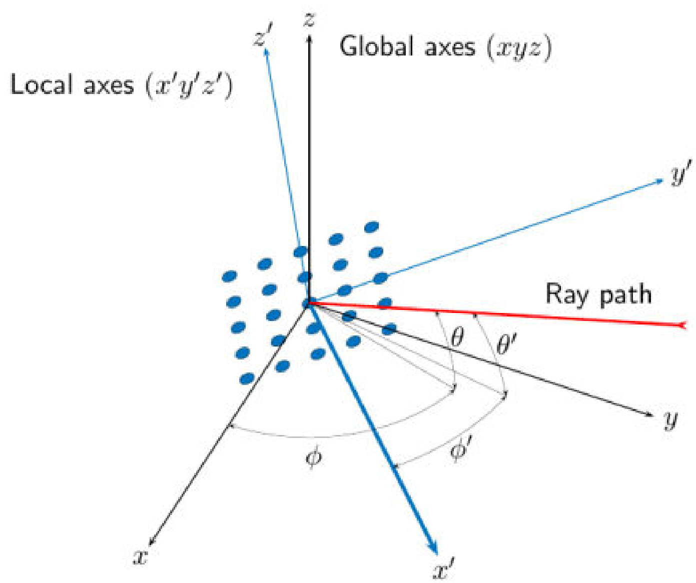
- When `model` is set to `"freespace"`, `ang` is a 2-by-*N* matrix and represents the angle of the path from a source point to a reference point.

- When `model` is set to `"two-ray"`, `ang` is a 2-by-2N matrix. Alternate columns of `ang` refer to the line-of-sight path and reflected path, respectively.

More About

Angles in Local and Global Coordinate Systems

The `rangeangle` function returns the path distance and path angles in either the global or local coordinate systems. Every antenna or microphone element and array has a gain pattern that is expressed in local angular coordinates of azimuth and elevation. As the element or array moves or rotates, the gain pattern is carried with it. To determine the strength of a signal, you must know the angle that the signal path makes with respect to the local angular coordinates of the element or array. By default, the `rangeangle` function determines the angle a signal path makes with respect to global coordinates. If you add the `refaxes` argument, you can compute the angles with respect to local coordinates. As an illustration, this figure shows a 5-by-5 uniform rectangular array (URA) rotated from the global coordinates (*xyz*) using `refaxes`. The *x*' axis of the local coordinate system (*x*'*y*'*z*') is aligned with the main axis of the array and moves as the array moves. The path length is independent of orientation. The global coordinate system defines the azimuth and elevation angles (Φ,θ) and the local coordinate system defines the azimuth and elevations angles (Φ',θ').



Local and Global Coordinate Axes

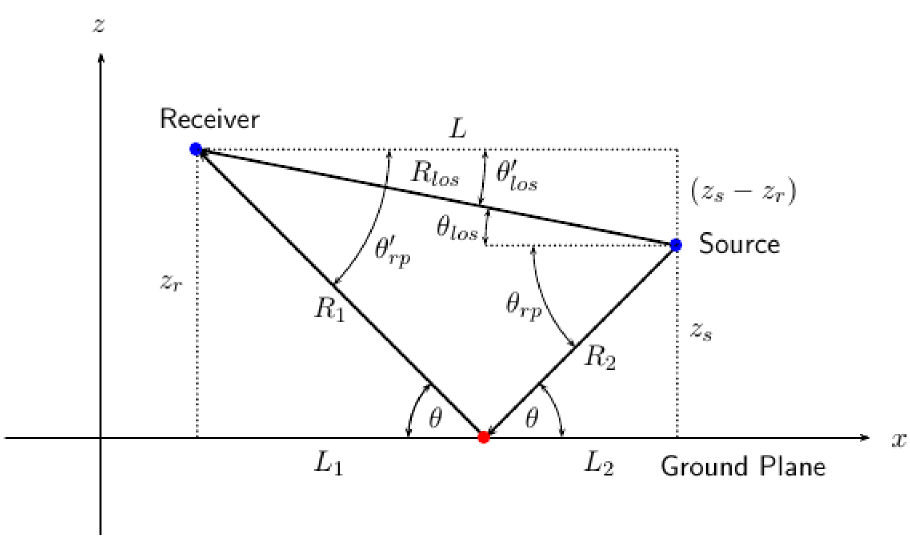
Free Space Propagation Model

The free-space signal propagation model states that a signal propagating from one point to another in a homogeneous, isotropic medium travels in a straight line, called the line-of-sight or direct path. The straight line is defined by the geometric vector from the radiation source to the destination. Similar assumptions are made for sonar but the term isovelocity channel is used in place of free space.

Two-Ray Propagation Model

A two-ray propagation channel is the next step up in complexity from a free-space channel and is the simplest case of a multipath propagation environment. The free-space channel models a straight-line *line-of-sight* path from point 1 to point 2. In a two-ray channel, the medium is specified as a homogeneous, isotropic medium with a reflecting planar boundary. The boundary is always set at $z = 0$. There are at most two rays propagating from point 1 to point 2. The first ray path propagates along the same line-of-sight path as in the free-space channel (see the `phased.FreeSpace` System object). The line-of-sight path is often called the *direct path*. The second ray reflects off the boundary before propagating to point 2. According to the Law of Reflection , the angle of reflection equals the angle of incidence. In short-range simulations such as cellular communications systems and automotive radars, you can assume that the reflecting surface, the ground or ocean surface, is flat.

The figure illustrates two propagation paths. From the source position, s_s, and the receiver position, s_r, you can compute the arrival angles of both paths, θ'_{los} and θ'_{rp}. The arrival angles are the elevation and azimuth angles of the arriving radiation with respect to a local coordinate system. In this case, the local coordinate system coincides with the global coordinate system. You can also compute the transmitting angles, θ_{los} and θ_{rp}. In the global coordinates, the angle of reflection at the boundary is the same as the angles θ_{rp} and θ'_{rp}. The reflection angle is important to know when you use angle-dependent reflection-loss data. You can determine the reflection angle by using the `rangeangle` function and setting the reference axes to the global coordinate system. The total path length for the line-of-sight path is shown in the figure by R_{los} which is equal to the geometric distance between source and receiver. The total path length for the reflected path is $R_{rp}= R_1 + R_2$. The quantity L is the ground range between source and receiver.



You can easily derive exact formulas for path lengths and angles in terms of the ground range and object heights in the global coordinate system.

$$\vec{R} = \vec{x}_s - \vec{x}_r$$

$$R_{los} = \left|\vec{R}\right| = \sqrt{(z_r - z_s)^2 + L^2}$$

$$R_1 = \frac{z_r}{z_r + z_z}\sqrt{(z_r + z_s)^2 + L^2}$$

$$R_2 = \frac{z_s}{z_s + z_r}\sqrt{(z_r + z_s)^2 + L^2}$$

$$R_{rp} = R_1 + R_2 = \sqrt{(z_r + z_s)^2 + L^2}$$

$$\tan\theta_{los} = \frac{(z_s - z_r)}{L}$$

$$\tan\theta_{rp} = -\frac{(z_s + z_r)}{L}$$

$$\theta'_{los} = -\theta_{los}$$

$$\theta'_{rp} = \theta_{rp}$$

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`global2localcoord` | `local2globalcoord` | `azel2uv` | `azel2phitheta`

Topics

“Global and Local Coordinate Systems”

rdcoupling

Range Doppler coupling

Syntax

```
dr = rdcoupling(fd,slope)
dr = rdcoupling(fd,slope,c)
```

Description

`dr = rdcoupling(fd,slope)` returns the range offset on page 2-384 due to the Doppler shift in a linear frequency modulated signal. For example, the signal can be a linear FM pulse or an FMCW signal. `slope` is the slope of the linear frequency modulation.

`dr = rdcoupling(fd,slope,c)` specifies the signal propagation speed.

Examples

Target Range After Correcting for Doppler Shift

Calculate the true range of the target for an FMCW waveform that sweeps a band of 30 MHz in 2 ms. The dechirped target echo has a beat frequency of 1 kHz. The processing of the target return indicates a Doppler shift of 100 Hz.

```
slope = 30e6/2e-3;
fb = 1e3;
fd = 100;
r = beat2range(fb,slope) - rdcoupling(fd,slope)
```

```
r = 10.9924
```

Input Arguments

`fd` — Doppler shift
array of real numbers

Doppler shift, specified as an array of real numbers.

Data Types: `double`

`slope` — Slope of linear frequency modulation
nonzero scalar

Slope of linear frequency modulation, specified as a nonzero scalar in hertz per second.

Data Types: `double`

`c` — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar in meters per second.

Data Types: `double`

Output Arguments

`dr` — Range offset due to Doppler shift
real scalar

Range offset due to Doppler shift, returned as an array of real numbers. The dimensions of `dr` match the dimensions of `fd`.

More About

Range Offset

The range offset is the difference between the estimated range and the true range. The difference arises from coupling between the range and Doppler shift.

Algorithms

The function computes `-c*fd/(2*slope)`.

Version History

Introduced in R2012b

References

[1] Barton, David K. *Radar System Analysis and Modeling*. Boston: Artech House, 2005.

[2] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`beat2range` | `dechirp` | `range2beat` | `stretchfreq2rng` | `phased.FMCWWaveform` | `phased.LinearFMWaveform`

Topics

"Automotive Adaptive Cruise Control Using FMCW Technology" (Radar Toolbox)

rocpfa

Receiver operating characteristic curves by false-alarm probability

Syntax

```
[Pd,SNR] = rocpfa(Pfa)
[Pd,SNR] = rocpfa(Pfa,Name=Value)
rocpfa(...)
```

Description

`[Pd,SNR] = rocpfa(Pfa)` returns the single-pulse detection probabilities, `Pd`, and required SNR values, `SNR`, for the false-alarm probabilities in the row or column vector `Pfa`. By default, for each false-alarm probability, the detection probabilities are computed for 101 equally spaced SNR values between 0 and 20 dB. The ROC curve is constructed assuming a single pulse in coherent receiver with a nonfluctuating target.

`[Pd,SNR] = rocpfa(Pfa,Name=Value)` returns detection probabilities and SNR values with additional options specified by one or more name-value arguments.

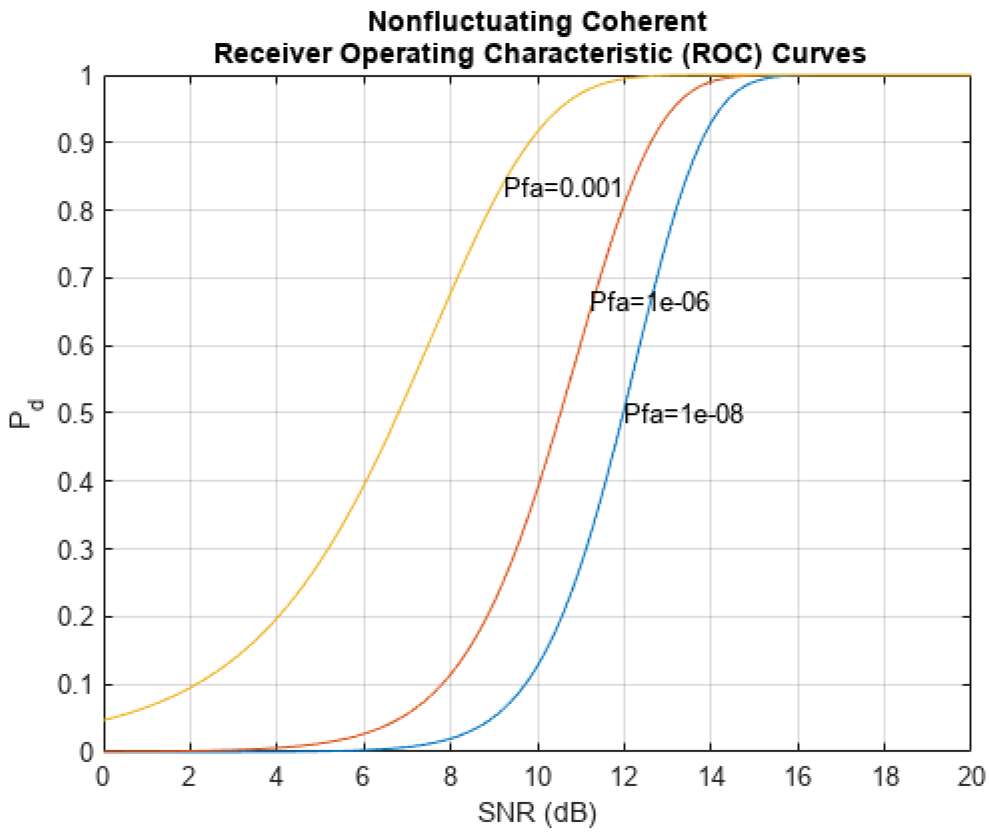
`rocpfa(...)` plots the ROC curves.

Examples

Plot ROC Curves for Different PFAs

Plot ROC curves for false-alarm probabilities of `1e-8`, `1e-6`, and `1e-3`, assuming no pulse integration.

```
Pfa = [1e-8 1e-6 1e-3];
rocpfa(Pfa,SignalType="NonfluctuatingCoherent")
```



Input Arguments

`Pfa` — False-alarm probabilities

vector

False-alarm probabilities, specified as a row or column vector.

Example: `[1e-8 1e-6 1e-3]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MaxSNR=15,NumPoints=64,NumPulses=10`

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MaxSNR',15,'NumPoints',64,'NumPulses',10`

`MaxSNR` — Maximum SNR to include in the ROC calculation

`20` (default) | positive scalar

Maximum SNR to include in the ROC calculation, specified as a positive scalar.

Data Types: `double`

`MinSNR` — Minimum SNR to include in the ROC calculation
`0` (default) | positive scalar

Minimum SNR to include in the ROC calculation, specified as a positive scalar.

Data Types: `double`

`NumPoints` — Number of SNR values to use when calculating the ROC curves
`101` (default) | positive integer

Number of SNR values to use when calculating the ROC curves, specified as a positive integer. The actual values are equally spaced between `MinSNR` and `MaxSNR`.

Data Types: `double`

`NumPulses` — Number of pulses to integrate
`1` (default) | positive integer

Number of pulses to integrate when calculating the ROC curves, specified as a positive integer. A value of `1` indicates no pulse integration.

Data Types: `double`

`SignalType` — Type of received signal
`"NonfluctuatingCoherent"` (default) | `"NonfluctuatingNoncoherent"` | `"Real"` | `"Swerling1"` | `"Swerling2"` | `"Swerling3"` | `"Swerling4"`

This property specifies the type of received signal or, equivalently, the probability density functions (PDF) used to compute the ROC. Valid values are: `"Real"`, `"NonfluctuatingCoherent"`, `"NonfluctuatingNoncoherent"`, `"Swerling1"`, `"Swerling2"`, `"Swerling3"`, and `"Swerling4"`. Values are not case sensitive.

The `"NonfluctuatingCoherent"` signal type assumes that the noise in the received signal is a complex-valued, Gaussian random variable. This variable has independent zero-mean real and imaginary parts each with variance $\sigma^2/2$ under the null hypothesis. In the case of a single pulse in a coherent receiver with complex white Gaussian noise, the probability of detection, P_D, for a given false-alarm probability, P_{FA} is:

$$P_D = \frac{1}{2}\text{erfc}(\text{erfc}^{-1}(2P_{FA}) - \sqrt{\chi})$$

where `erfc` and `erfc`$^{-1}$ are the complementary error function and that function's inverse, and χ is the SNR not expressed in decibels.

For details about the other supported signal types, see [1] on page 2-388.

Data Types: `char` | `string`

Output Arguments

Pd — Detection probabilities
vector

Detection probabilities corresponding to the false-alarm probabilities, returned as a vector. For each false-alarm probability in `Pfa`, `Pd` contains one column of detection probabilities.

SNR — Signal-to-noise ratios
column vector

Signal-to-noise ratios, returned as a column vector. By default, the SNR values are 101 equally spaced values between 0 and 20. To change the range of SNR values, use the optional `MinSNR` or `MaxSNR` input argument. To change the number of SNR values, use the optional `NumPoints` input argument.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.
- Supported only when output arguments are specified.

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005, pp 298–336.

See Also

`npwgnthresh` | `rocsnr` | `shnidman`

rocsnr

Receiver operating characteristic curves by SNR

Syntax

```
[Pd,Pfa] = rocsnr(SNRdB)
[Pd,Pfa] = rocsnr(SNRdB,Name=Value)
rocsnr( ___ )
```

Description

`[Pd,Pfa] = rocsnr(SNRdB)` returns the single-pulse detection probabilities, `Pd`, and false-alarm probabilities, `Pfa`, for the SNRs in the vector `SNRdB`. By default, for each SNR, the detection probabilities are computed for 101 false-alarm probabilities between `1e–10` and `1`. The false-alarm probabilities are logarithmically equally spaced. The ROC curve is constructed assuming a coherent receiver with a nonfluctuating target.

`[Pd,Pfa] = rocsnr(SNRdB,Name=Value)` returns detection probabilities and false-alarm probabilities with additional options specified by one or more name-value arguments.

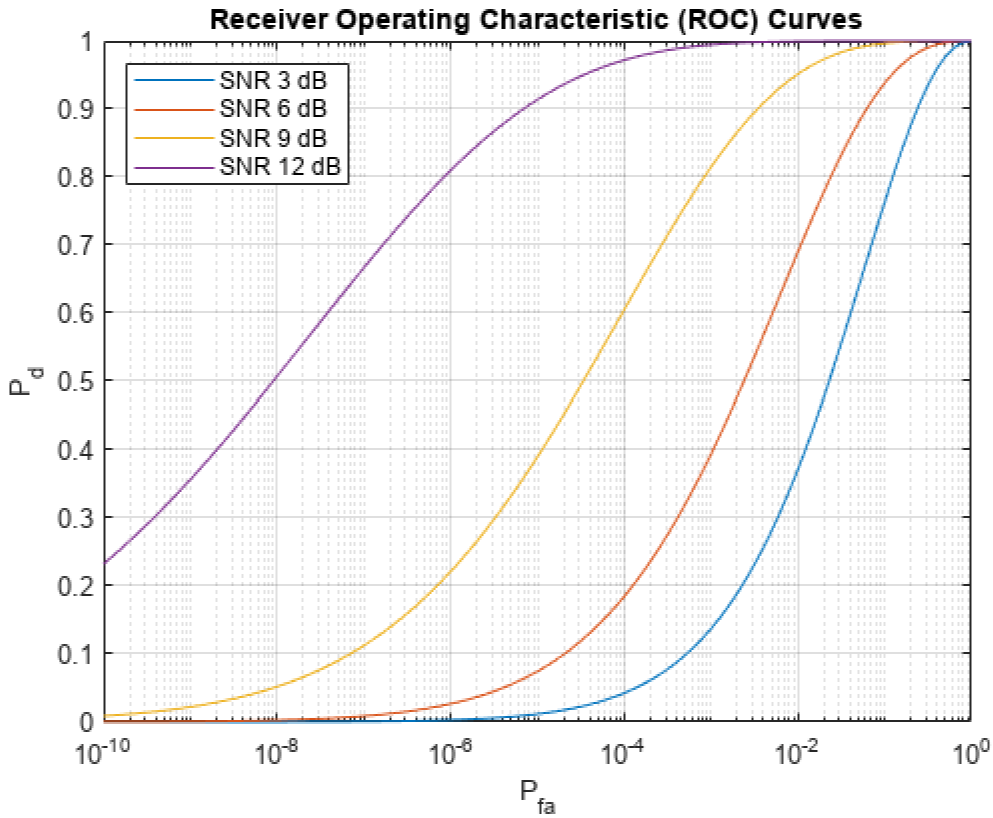
`rocsnr( ___ )` plots the ROC curves.

Examples

ROC Curves for Different SNRs

Plot ROC curves for different SNRs for a single pulse.

```
SNRdB = [3 6 9 12];

[Pd,Pfa] = rocsnr(SNRdB,SignalType="NonfluctuatingCoherent");

semilogx(Pfa,Pd)
grid on
xlabel("P_{fa}")
ylabel("P_d")
legend("SNR "+SNRdB+" dB",Location="northwest")
title("Receiver Operating Characteristic (ROC) Curves")
```



Input Arguments

SNRdB — Signal-to-noise ratios
vector

Signal-to-noise ratios in decibels, specified as a row or column vector.

Example: `[3 6 9 12]`

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Example: `MinPfa=1e-8,NumPoints=64,NumPulses=10`

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'MinPfa',1e-8,'NumPoints',64,'NumPulses',10`

MaxPfa — Maximum false-alarm probability to include in the ROC calculation
`1` (default) | positive scalar

Maximum false-alarm probability to include in the ROC calculation, specified as a positive scalar.

Data Types: `double`

`MinPfa` — Minimum false-alarm probability to include in the ROC calculation
`1e-10` (default) | positive scalar

Minimum false-alarm probability to include in the ROC calculation, specified as a positive scalar.

Data Types: `double`

`NumPulses` — Number of pulses to integrate
`1` (default) | positive integer

Number of pulses to integrate when calculating the ROC curves, specified as a positive integer. A value of `1` indicates no pulse integration.

Data Types: `double`

`NumPoints` — Number of SNR values to use when calculating the ROC curves
`101` (default) | positive integer

Number of SNR values to use when calculating the ROC curves, specified as a positive integer. The actual values are equally spaced between `MinSNR` and `MaxSNR`.

Data Types: `double`

`SignalType` — Type of received signal
`"NonfluctuatingCoherent"` (default) | `"NonfluctuatingNoncoherent"` | `"Real"` | `"Swerling1"` | `"Swerling2"` | `"Swerling3"` | `"Swerling4"`

This property specifies the type of received signal or, equivalently, the probability density functions (PDF) used to compute the ROC. Valid values are: `"Real"`, `"NonfluctuatingCoherent"`, `"NonfluctuatingNoncoherent"`, `"Swerling1"`, `"Swerling2"`, `"Swerling3"`, and `"Swerling4"`. Values are not case sensitive.

The `"NonfluctuatingCoherent"` signal type assumes that the noise in the received signal is a complex-valued, Gaussian random variable. This variable has independent zero-mean real and imaginary parts each with variance $\sigma^2/2$ under the null hypothesis. In the case of a single pulse in a coherent receiver with complex white Gaussian noise, the probability of detection, P_D, for a given false-alarm probability, P_{FA} is:

$$P_D = \frac{1}{2}\mathrm{erfc}(\mathrm{erfc}^{-1}(2P_{FA}) - \sqrt{\chi})$$

where `erfc` and `erfc`$^{-1}$ are the complementary error function and that function's inverse, and χ is the SNR not expressed in decibels.

For details about the other supported signal types, see [1].

Data Types: `char` | `string`

Output Arguments

`Pd` — Detection probabilities
vector

Detection probabilities corresponding to the false-alarm probabilities, returned as a vector. For each SNR in `SNRdB`, `Pd` contains one column of detection probabilities.

`Pfa` — False-alarm probabilities
column vector

False-alarm probabilities, returned as a column vector. By default, the false-alarm probabilities are 101 logarithmically equally spaced values between `1e–10` and `1`. To change the range of probabilities, use the optional `MinPfa` or `MaxPfa` input argument. To change the number of probabilities, use the optional `NumPoints` input argument.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.
- Supported only when output arguments are specified.

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005, pp 298–336.

See Also

`npwgnthresh` | `rocpfa` | `shnidman`

rootmusicdoa

Direction of arrival using Root MUSIC

Syntax

```
ang = rootmusicdoa(R,nsig)
ang = rootmusicdoa( ___ ,'Name','Value')
```

Description

`ang = rootmusicdoa(R,nsig)` estimates the directions of arrival, `ang`, of a set of plane waves received on a uniform line array (ULA). The estimation uses the root MUSIC algorithm. The input arguments are the estimated spatial covariance matrix between sensor elements, `R`, and the number of arriving signals, `nsig`. In this syntax, sensor elements are spaced one-half wavelength apart.

`ang = rootmusicdoa( ___ ,'Name','Value')` allows you to specify additional input parameters in the form of Name-Value pairs. This syntax can use any of the input arguments in the previous syntax.

Examples

Three Signals Arriving at Half-Wavelength-Spaced ULA

Assume a half-wavelength spaced uniform line array with 10 elements. Three plane waves arrive from the 0°, –25°, and 30° azimuth directions. Elevation angles are 0°. The noise is spatially and temporally white Gaussian noise.

Set the SNR for each signal to 5 dB. Find the arrival angles.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25 30];
Nsig = 3;
R = sensorcov(elementPos,angles,db2pow(-5));
doa = rootmusicdoa(R,Nsig)
```

```
doa = 1×3

    0.0000   30.0000  -25.0000
```

These angles agree with the known input angles.

Three Signals Arriving at 0.4-Wavelength-Spaced ULA

Assume a uniform line array 10 elements, as in the previous example. But now the element spacing is smaller than one-half wavelength. Three plane waves arrive from the 0°, –25°, and 30° azimuth directions. Elevation angles are 0°. The noise is spatially and temporally white Gaussian noise. The SNR for each signal is 5 dB.

Set element spacing to 0.4 wavelengths using the `ElementSpacing` name-value pair. Then, find the arrival angles.

```
N = 10;
d = 0.4;
elementPos = (0:N-1)*d;
angles = [0 -25 30];
Nsig = 3;
R = sensorcov(elementPos,angles,db2pow(-5));
doa = rootmusicdoa(R,Nsig,'ElementSpacing',d)
```

```
doa = 1×3

  -25.0000   30.0000    0.0000
```

The solution agrees with the known angles.

Input Arguments

R — Spatial covariance matrix
complex-valued positive-definite *N*-by-*N* matrix

Spatial covariance matrix, specified as a complex-valued, positive-definite, *N*-by-*N* matrix. In this matrix, *N* represents the number of elements in the ULA array. If `R` is not Hermitian, a Hermitian matrix is formed by averaging the matrix and its conjugate transpose, `(R+R')/2`.

Example: [4.3162, –0.2777 –0.2337i; –0.2777 + 0.2337i , 4.3162]

Data Types: `double`
Complex Number Support: Yes

`nsig` — Number of arriving signals
positive integer

Number of arriving signals, specified as a positive integer. The number of signals must be smaller than the number of elements in the ULA array.

Example: 2

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: 'ElementSpacing', 0.4

`ElementSpacing` — ULA element spacing
0.5 (default) | real-valued positive scalar

ULA element spacing, specified as a real-valued, positive scalar. Position units are measured in terms of signal wavelength.

Example: 0.4

Data Types: `double`

Output Arguments

ang — Directions of arrival angles
real-valued 1-by-*M* row vector

Directions of arrival angle, returned as a real-valued, 1-by-*M* vector. The dimension *M* is the number of arriving signals specified in the argument `nsig`. Angle units are degrees and angle values lie between –90° and 90°.

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`aictest` | `espritdoa` | `spsmooth` | `phased.RootMUSICEstimator`

rotx

Rotation matrix for rotations around x-axis

Syntax

```
R = rotx(ang)
```

Description

`R = rotx(ang)` creates a 3-by-3 matrix for rotating a 3-by-1 vector or 3-by-N matrix of vectors around the x-axis by `ang` degrees. When acting on a matrix, each column of the matrix represents a different vector. For the rotation matrix `R` and vector `v`, the rotated vector is given by `R*v`.

Examples

Rotation Matrix for 30° Rotation

Construct the matrix for a rotation of a vector around the x-axis by 30°. Then let the matrix operate on a vector.

```
R = rotx(30)
```

```
R = 3×3

    1.0000         0         0
         0    0.8660   -0.5000
         0    0.5000    0.8660
```

```
x = [2;-2;4];
y = R*x
```

```
y = 3×1

    2.0000
   -3.7321
    2.4641
```

Under a rotation around the *x*-axis, the x-component of a vector is invariant.

Input Arguments

ang — Rotation angle
real-valued scalar

Rotation angle specified as a real-valued scalar. The rotation angle is positive if the rotation is in the counter-clockwise direction when viewed by an observer looking along the x-axis towards the origin. Angle units are in degrees.

Example: 30.0

Data Types: `double`

Output Arguments

R — Rotation matrix
real-valued orthogonal matrix

3-by-3 rotation matrix returned as

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

for a rotation angle α.

More About

Rotation Matrices

Rotation matrices are used to rotate a vector into a new direction.

In transforming vectors in three-dimensional space, rotation matrices are often encountered. Rotation matrices are used in two senses: they can be used to rotate a vector into a new position or they can be used to rotate a coordinate basis (or coordinate system) into a new one. In this case, the vector is left alone but its components in the new basis will be different from those in the original basis. In Euclidean space, there are three basic rotations: one each around the x, y and z axes. Each rotation is specified by an angle of rotation. The rotation angle is defined to be positive for a rotation that is counterclockwise when viewed by an observer looking along the rotation axis towards the origin. Any arbitrary rotation can be composed of a combination of these three *(Euler's rotation theorem)*. For example, you can rotate a vector in any direction using a sequence of three rotations: $\mathbf{v}' = A\mathbf{v} = R_z(\gamma)R_y(\beta)R_x(\alpha)\mathbf{v}$.

The rotation matrices that rotate a vector around the x, y, and z-axes are given by:

- Counterclockwise rotation around x-axis

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

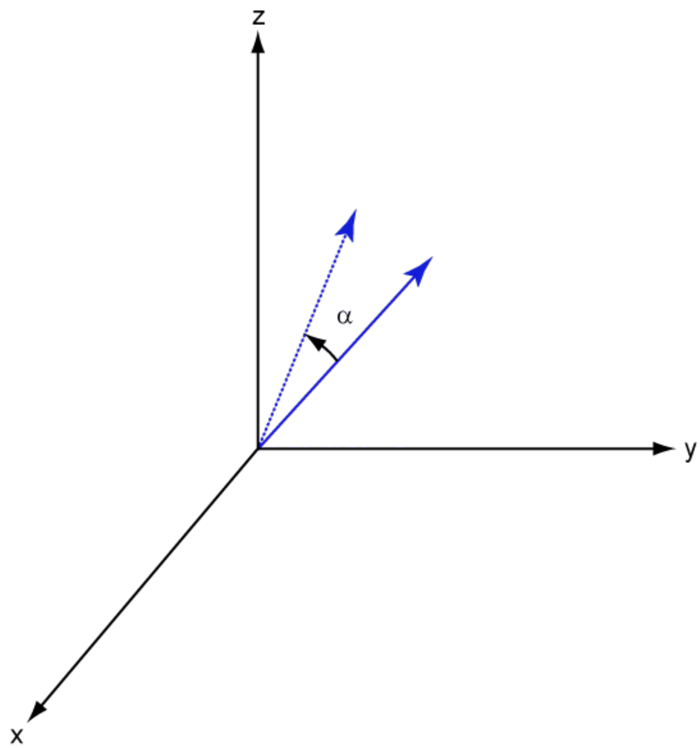
- Counterclockwise rotation around y-axis

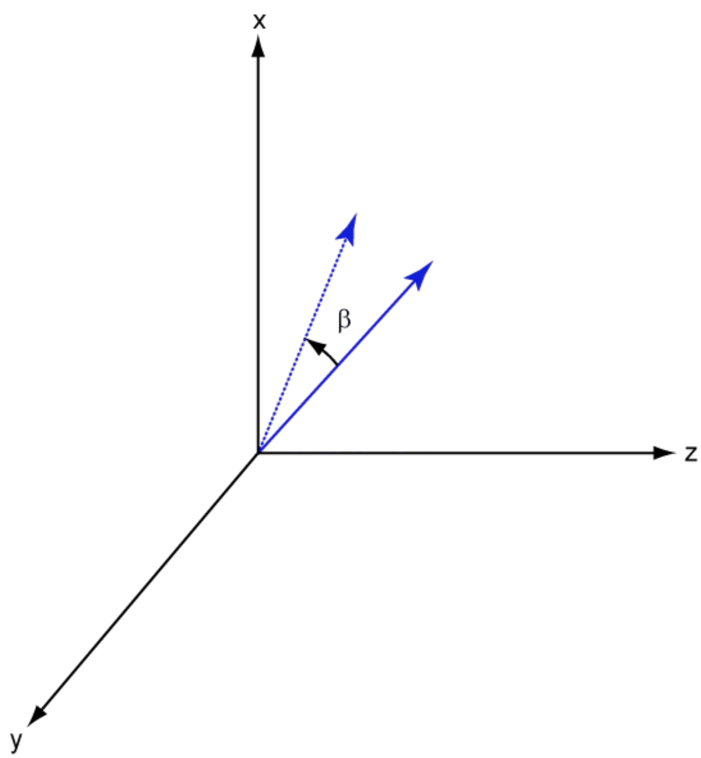
$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

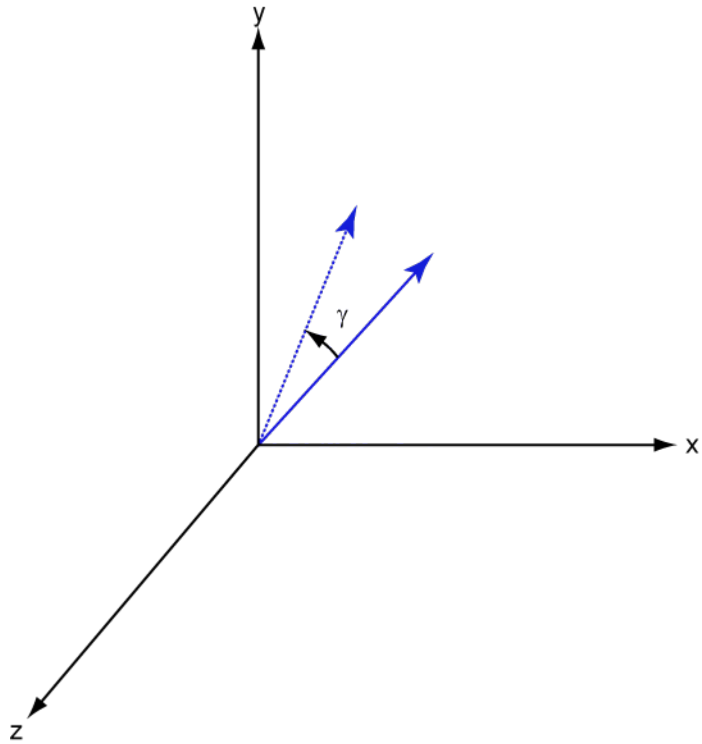
- Counterclockwise rotation around z-axis

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The following three figures show what positive rotations look like for each rotation axis:







For any rotation, there is an inverse rotation satisfying $A^{-1}A = 1$. For example, the inverse of the x-axis rotation matrix is obtained by changing the sign of the angle:

$$R_x^{-1}(\alpha) = R_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} = R_x'(\alpha)$$

This example illustrates a basic property: the inverse rotation matrix is the transpose of the original. Rotation matrices satisfy *A'A = 1*, and consequently *det(A) = 1*. Under rotations, vector lengths are preserved as well as the angles between vectors.

We can think of rotations in another way. Consider the original set of basis vectors, **i**, **j**, **k**, and rotate them all using the rotation matrix *A*. This produces a new set of basis vectors **i**′, **j**,′ **k**′ related to the original by:

$$\begin{aligned} \mathbf{i}' &= A\mathbf{i} \\ \mathbf{j}' &= A\mathbf{j} \\ \mathbf{k}' &= A\mathbf{k} \end{aligned}$$

Using the transpose, you can write the new basis vectors as a linear combinations of the old basis vectors:

$$\begin{bmatrix} \mathbf{i}' \\ \mathbf{j}' \\ \mathbf{k}' \end{bmatrix} = A' \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}$$

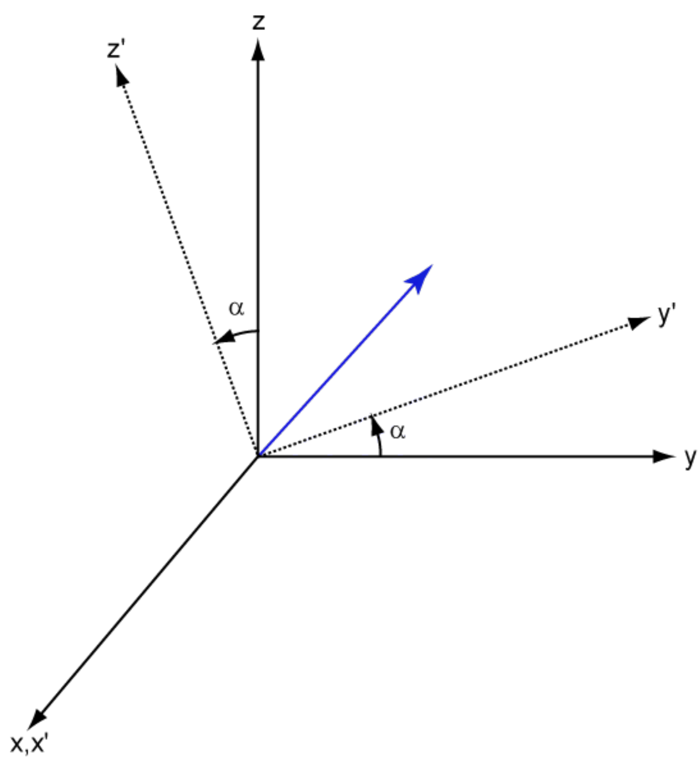
Now any vector can be written as a linear combination of either set of basis vectors:

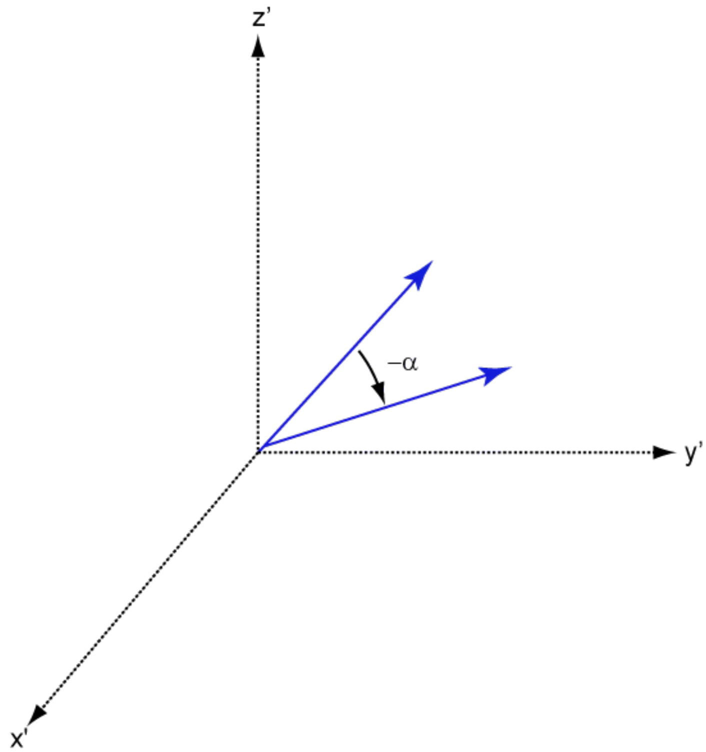
$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k} = v'_x\mathbf{i}' + v'_y\mathbf{j}' + v'_z\mathbf{k}'$$

Using algebraic manipulation, you can derive the transformation of components for a fixed vector when the basis (or coordinate system) rotates. This transformation uses the transpose of the rotation matrix.

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = A^{-1}\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = A'\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

The next figure illustrates how a vector is transformed as the coordinate system rotates around the x-axis. The figure after shows how this transformation can be interpreted as a rotation *of the vector* in the opposite direction.





Version History

Introduced in R2013a

References

[1] Goldstein, H., C. Poole and J. Safko, *Classical Mechanics*, 3rd Edition, San Francisco: Addison Wesley, 2002, pp. 142–144.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`roty` | `rotz`

roty

Rotation matrix for rotations around y-axis

Syntax

```
R = roty(ang)
```

Description

`R = roty(ang)` creates a 3-by-3 matrix used to rotate a 3-by-1 vector or 3-by-*N* matrix of vectors around the *y*-axis by `ang` degrees. When acting on a matrix, each column of the matrix represents a different vector. For the rotation matrix *R* and vector *v*, the rotated vector is given by *R*v*.

Examples

Rotation Matrix for 45° Rotation

Construct the matrix for a rotation of a vector around the y-axis by 45°. Then let the matrix operate on a vector.

```
R = roty(45)
```

```
R = 3×3

    0.7071         0    0.7071
         0    1.0000         0
   -0.7071         0    0.7071
```

```
v = [1;-2;4];
y = R*v
```

```
y = 3×1

    3.5355
   -2.0000
    2.1213
```

Under a rotation around the y-axis, the *y*-component of a vector is invariant.

Input Arguments

ang — Rotation angle
real-valued scalar

Rotation angle specified as a real-valued scalar. The rotation angle is positive if the rotation is in the counter-clockwise direction when viewed by an observer looking along the y-axis towards the origin. Angle units are in degrees.

Example: 30.0

Data Types: `double`

Output Arguments

R — Rotation matrix
real-valued orthogonal matrix

3-by-3 rotation matrix returned as

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

for a rotation angle β.

More About

Rotation Matrices

Rotation matrices are used to rotate a vector into a new direction.

In transforming vectors in three-dimensional space, rotation matrices are often encountered. Rotation matrices are used in two senses: they can be used to rotate a vector into a new position or they can be used to rotate a coordinate basis (or coordinate system) into a new one. In this case, the vector is left alone but its components in the new basis will be different from those in the original basis. In Euclidean space, there are three basic rotations: one each around the x, y and z axes. Each rotation is specified by an angle of rotation. The rotation angle is defined to be positive for a rotation that is counterclockwise when viewed by an observer looking along the rotation axis towards the origin. Any arbitrary rotation can be composed of a combination of these three *(Euler's rotation theorem)*. For example, you can rotate a vector in any direction using a sequence of three rotations: $\mathbf{v}' = A\mathbf{v} = R_z(\gamma)R_y(\beta)R_x(\alpha)\mathbf{v}$.

The rotation matrices that rotate a vector around the x, y, and z-axes are given by:

- Counterclockwise rotation around x-axis

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

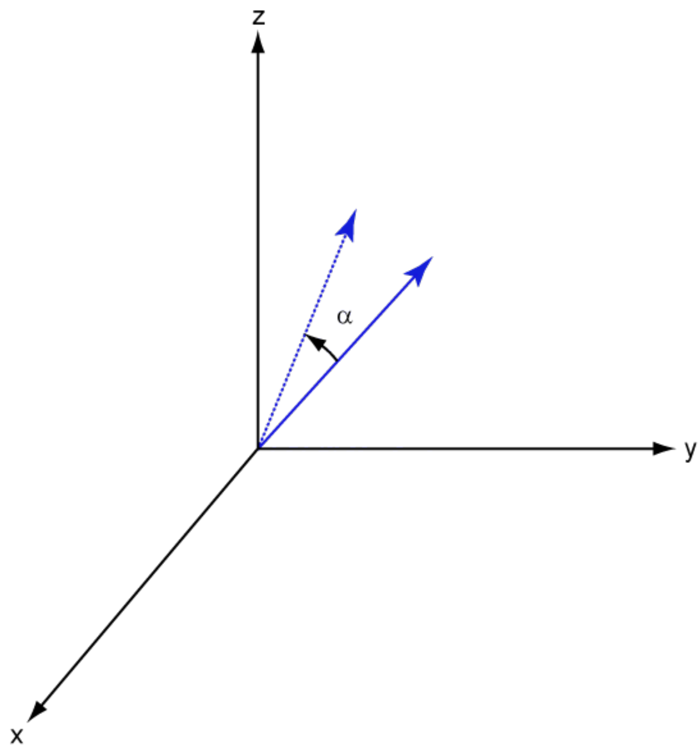
- Counterclockwise rotation around y-axis

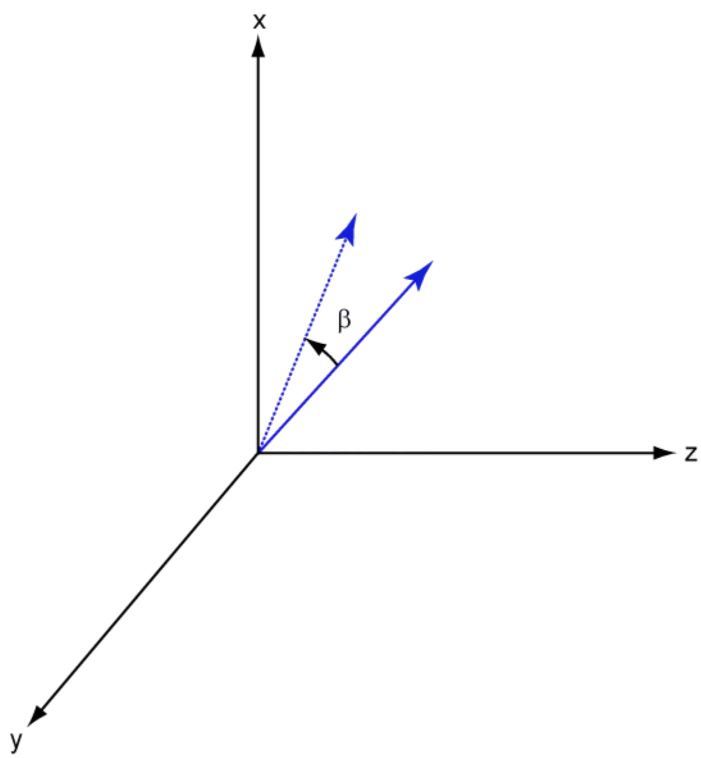
$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

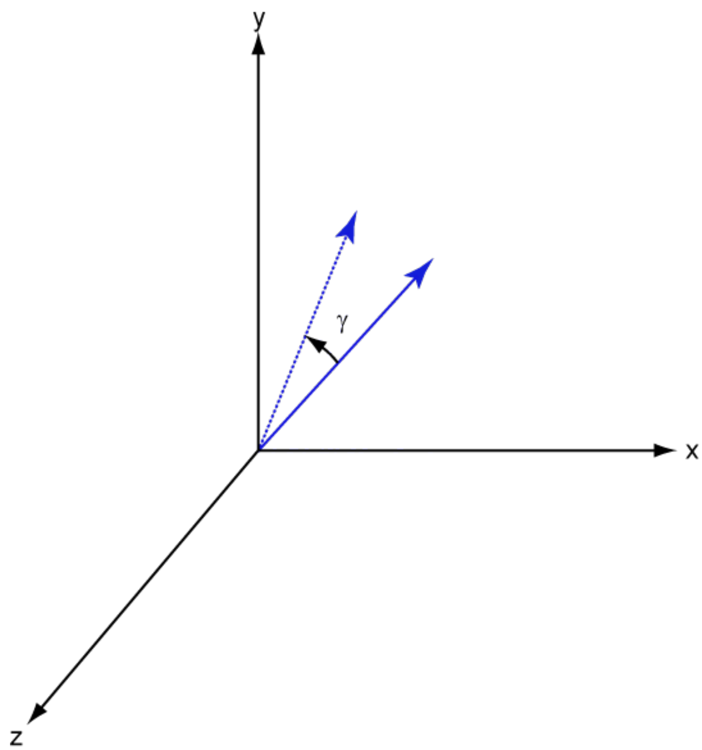
- Counterclockwise rotation around z-axis

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The following three figures show what positive rotations look like for each rotation axis:







For any rotation, there is an inverse rotation satisfying $A^{-1}A = 1$. For example, the inverse of the x-axis rotation matrix is obtained by changing the sign of the angle:

$$R_x^{-1}(\alpha) = R_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} = R_x'(\alpha)$$

This example illustrates a basic property: the inverse rotation matrix is the transpose of the original. Rotation matrices satisfy *A’A = 1*, and consequently *det(A) = 1*. Under rotations, vector lengths are preserved as well as the angles between vectors.

We can think of rotations in another way. Consider the original set of basis vectors, $\mathbf{i}, \mathbf{j}, \mathbf{k}$, and rotate them all using the rotation matrix A. This produces a new set of basis vectors $\mathbf{i}', \mathbf{j}',\mathbf{k}'$ related to the original by:

$$\mathbf{i}' = A\mathbf{i}$$
$$\mathbf{j}' = A\mathbf{j}$$
$$\mathbf{k}' = A\mathbf{k}$$

Using the transpose, you can write the new basis vectors as a linear combinations of the old basis vectors:

$$\begin{bmatrix} \mathbf{i}' \\ \mathbf{j}' \\ \mathbf{k}' \end{bmatrix} = A' \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}$$

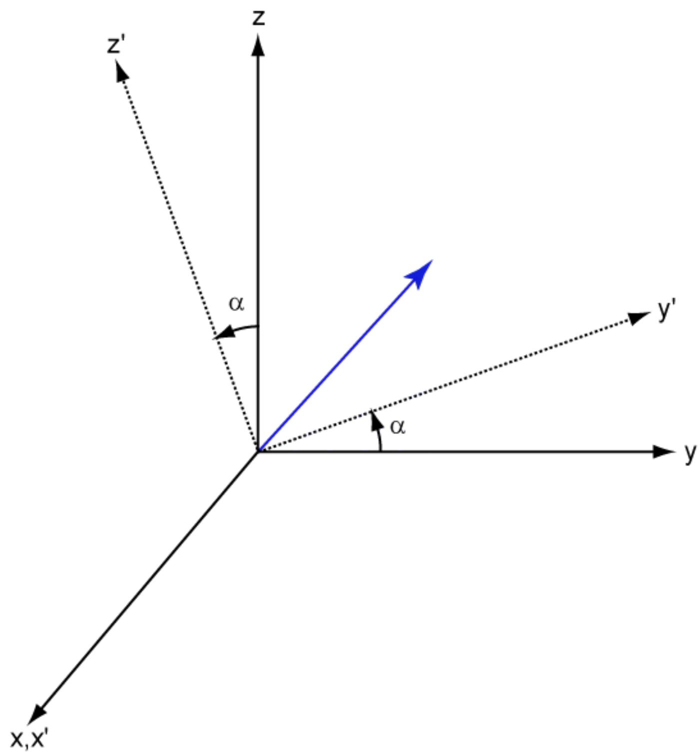
Now any vector can be written as a linear combination of either set of basis vectors:

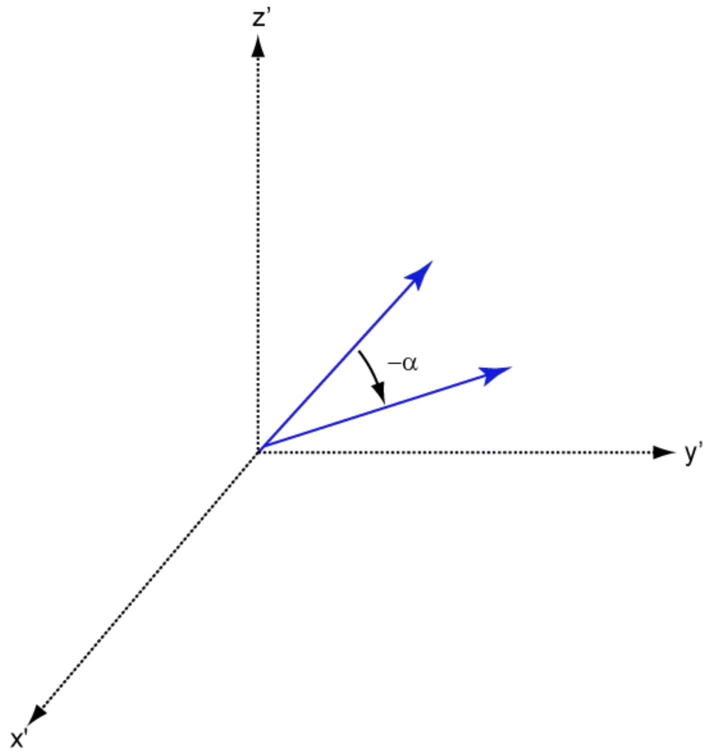
$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k} = v'_x\mathbf{i}' + v'_y\mathbf{j}' + v'_z\mathbf{k}'$$

Using algebraic manipulation, you can derive the transformation of components for a fixed vector when the basis (or coordinate system) rotates. This transformation uses the transpose of the rotation matrix.

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = A^{-1}\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = A'\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

The next figure illustrates how a vector is transformed as the coordinate system rotates around the x-axis. The figure after shows how this transformation can be interpreted as a rotation *of the vector* in the opposite direction.





Version History

Introduced in R2013a

References

[1] Goldstein, H., C. Poole and J. Safko, *Classical Mechanics*, 3rd Edition, San Francisco: Addison Wesley, 2002, pp. 142–144.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`rotx` | `rotz`

rotz

Rotation matrix for rotations around z-axis

Syntax

```
R = rotz(ang)
```

Description

`R = rotz(ang)` creates a 3-by-3 matrix used to rotate a 3-by-1 vector or 3-by-N matrix of vectors around the z-axis by `ang` degrees. When acting on a matrix, each column of the matrix represents a different vector. For the rotation matrix `R` and vector `v`, the rotated vector is given by `R*v`.

Examples

Rotation Matrix for 45° Rotation

Construct the matrix for the rotation of a vector around the z-axis by 45°. Then let the matrix operate on a vector.

```
R = rotz(45)
```

```
R = 3×3

    0.7071   -0.7071         0
    0.7071    0.7071         0
         0         0    1.0000
```

```
v = [1;-2;4];
y = R*v
```

```
y = 3×1

    2.1213
   -0.7071
    4.0000
```

Under a rotation around the z-axis, the *z*-component of a vector is invariant.

Input Arguments

ang — Rotation angle
real-valued scalar

Rotation angle specified as a real-valued scalar. The rotation angle is positive if the rotation is in the counter-clockwise direction when viewed by an observer looking along the z-axis towards the origin. Angle units are in degrees.

Example: 45.0

Data Types: `double`

Output Arguments

R — Rotation matrix
real-valued orthogonal matrix

3-by-3 rotation matrix returned as

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

for a rotation angle γ.

More About

Rotation Matrices

Rotation matrices are used to rotate a vector into a new direction.

In transforming vectors in three-dimensional space, rotation matrices are often encountered. Rotation matrices are used in two senses: they can be used to rotate a vector into a new position or they can be used to rotate a coordinate basis (or coordinate system) into a new one. In this case, the vector is left alone but its components in the new basis will be different from those in the original basis. In Euclidean space, there are three basic rotations: one each around the x, y and z axes. Each rotation is specified by an angle of rotation. The rotation angle is defined to be positive for a rotation that is counterclockwise when viewed by an observer looking along the rotation axis towards the origin. Any arbitrary rotation can be composed of a combination of these three *(Euler's rotation theorem)*. For example, you can rotate a vector in any direction using a sequence of three rotations: $\mathbf{v}' = A\mathbf{v} = R_z(\gamma)R_y(\beta)R_x(\alpha)\mathbf{v}$.

The rotation matrices that rotate a vector around the x, y, and z-axes are given by:

- Counterclockwise rotation around x-axis

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

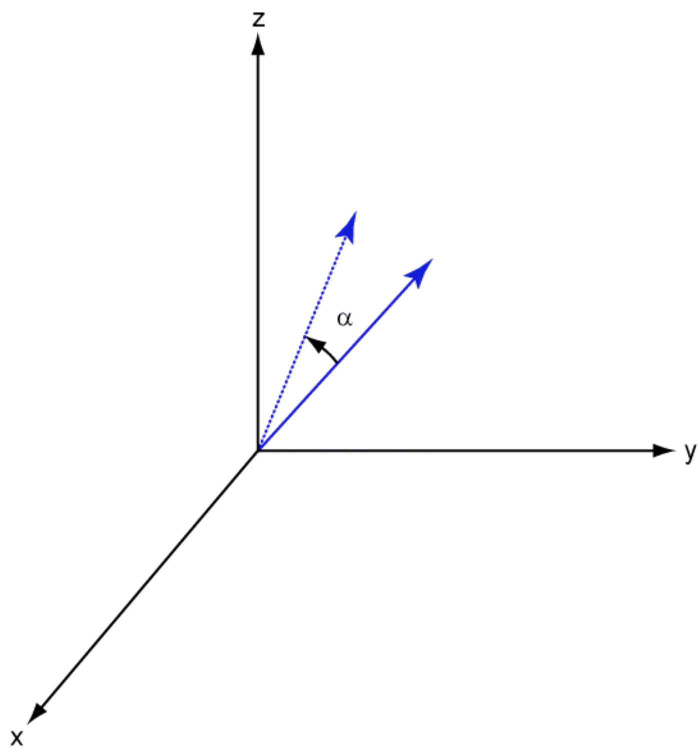
- Counterclockwise rotation around y-axis

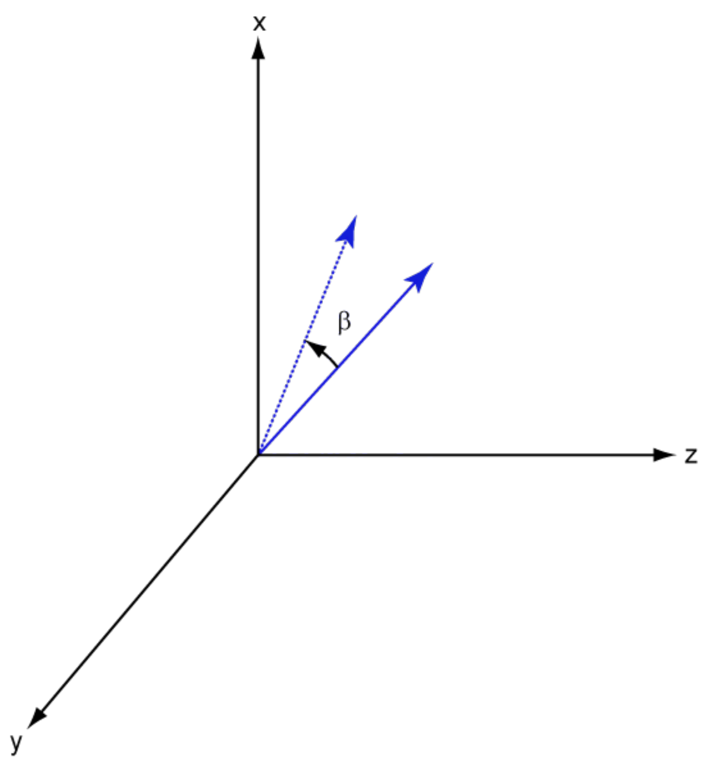
$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

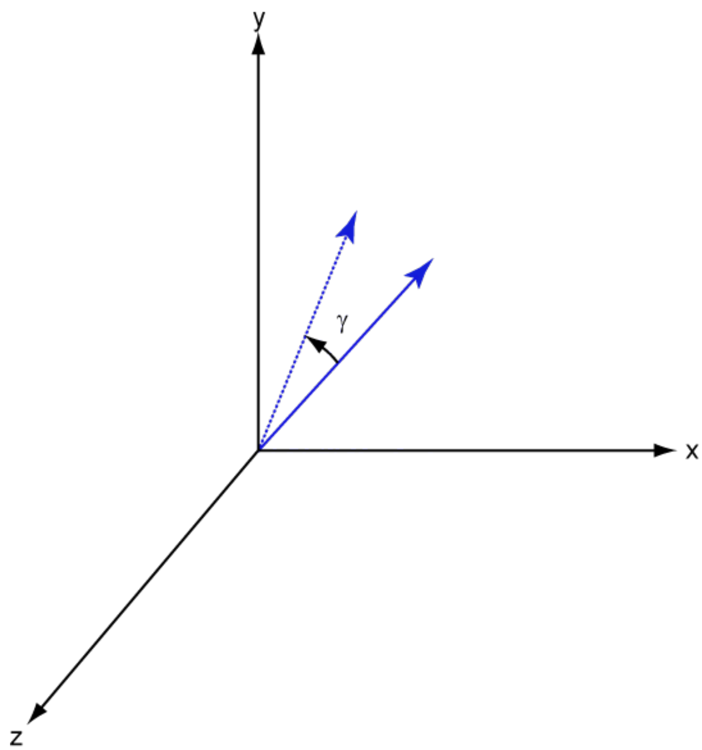
- Counterclockwise rotation around z-axis

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The following three figures show what positive rotations look like for each rotation axis:







For any rotation, there is an inverse rotation satisfying $A^{-1}A = 1$. For example, the inverse of the x-axis rotation matrix is obtained by changing the sign of the angle:

$$R_x^{-1}(\alpha) = R_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} = R_x'(\alpha)$$

This example illustrates a basic property: the inverse rotation matrix is the transpose of the original. Rotation matrices satisfy *A′A = 1*, and consequently *det(A) = 1*. Under rotations, vector lengths are preserved as well as the angles between vectors.

We can think of rotations in another way. Consider the original set of basis vectors, $\mathbf{i}, \mathbf{j}, \mathbf{k}$, and rotate them all using the rotation matrix A. This produces a new set of basis vectors $\mathbf{i}', \mathbf{j},' \mathbf{k}'$ related to the original by:

$$\begin{aligned} \mathbf{i}' &= A\mathbf{i} \\ \mathbf{j}' &= A\mathbf{j} \\ \mathbf{k}' &= A\mathbf{k} \end{aligned}$$

Using the transpose, you can write the new basis vectors as a linear combinations of the old basis vectors:

$$\begin{bmatrix} \mathbf{i}' \\ \mathbf{j}' \\ \mathbf{k}' \end{bmatrix} = A' \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}$$

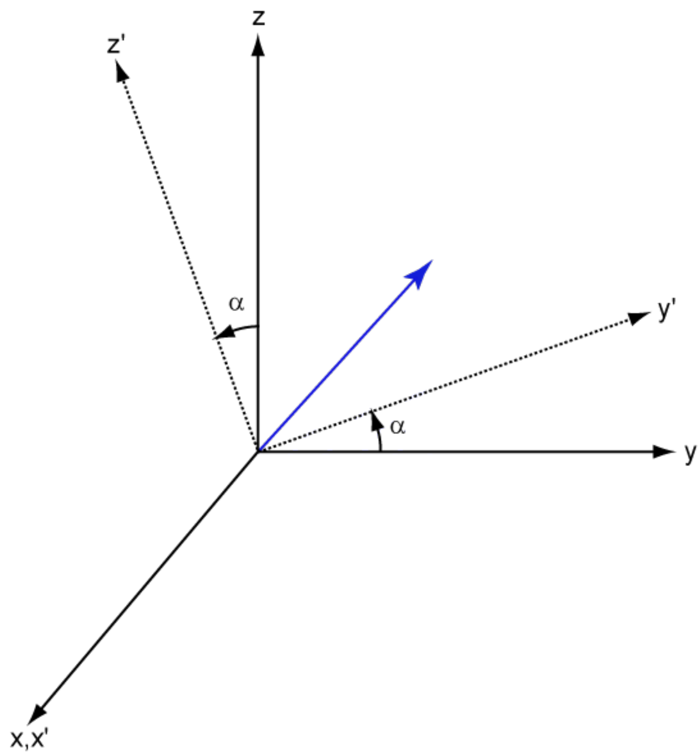
Now any vector can be written as a linear combination of either set of basis vectors:

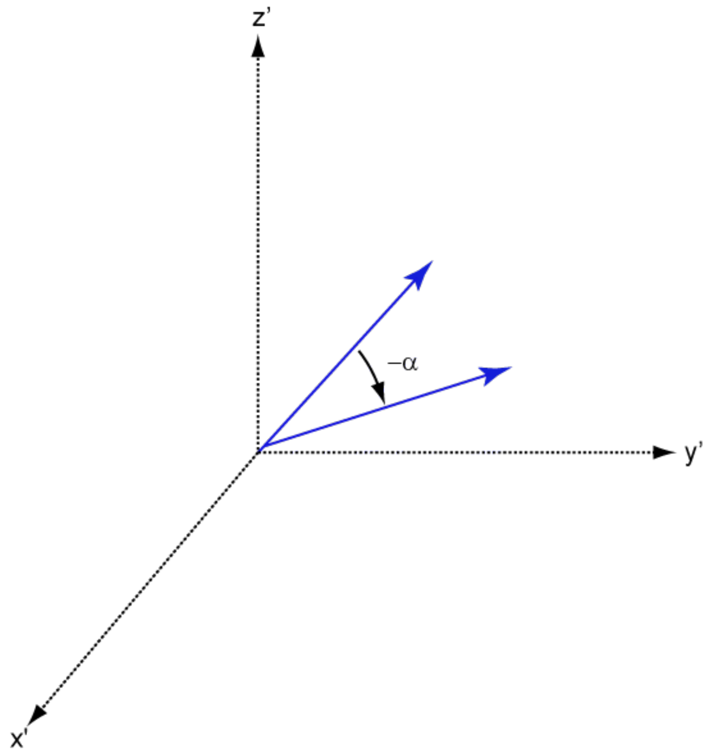
$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k} = v'_x\mathbf{i}' + v'_y\mathbf{j}' + v'_z\mathbf{k}'$$

Using algebraic manipulation, you can derive the transformation of components for a fixed vector when the basis (or coordinate system) rotates. This transformation uses the transpose of the rotation matrix.

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = A^{-1}\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = A'\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

The next figure illustrates how a vector is transformed as the coordinate system rotates around the x-axis. The figure after shows how this transformation can be interpreted as a rotation *of the vector* in the opposite direction.





Version History

Introduced in R2013a

References

[1] Goldstein, H., C. Poole and J. Safko, *Classical Mechanics*, 3rd Edition, San Francisco: Addison Wesley, 2002, pp. 142–144.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`rotx` | `roty`

sensorcov

Sensor spatial covariance matrix

Syntax

```
xcov = sensorcov(pos,ang)
xcov = sensorcov(pos,ang,ncov)
xcov = sensorcov(pos,ang,ncov,scov)
```

Description

`xcov = sensorcov(pos,ang)` returns the sensor spatial covariance matrix, `xcov`, for narrowband plane wave signals arriving at a sensor array. The sensor array is defined by the sensor positions specified in the `pos` argument. The signal arrival directions are specified by azimuth and elevation angles in the `ang` argument. In this syntax, the noise power is assumed to be zero at all sensors, and the signal power is assumed to be unity for all signals.

`xcov = sensorcov(pos,ang,ncov)` specifies, in addition, the spatial noise covariance matrix, `ncov`. This value represents the noise power on each sensor as well as the correlation of the noise between sensors. In this syntax, the signal power is assumed to be unity for all signals. This syntax can use any of the input arguments in the previous syntax.

`xcov = sensorcov(pos,ang,ncov,scov)` specifies, in addition, the signal covariance matrix, `scov`, which represents the power in each signal and the correlation between signals. This syntax can use any of the input arguments in the previous syntaxes.

Examples

Covariance Matrix for Two Signals Without Noise

Create a covariance matrix for a 3-element, half-wavelength-spaced uniform line array. Use the default syntax, which assumes no noise power and unit signal power.

```
N = 3;
d = 0.5;
elementPos = (0:N-1)*d;
xcov = sensorcov(elementPos,[30 60])
```

```
xcov = 3×3 complex

   2.0000 + 0.0000i  -0.9127 - 1.4086i  -0.3339 + 0.7458i
  -0.9127 + 1.4086i   2.0000 + 0.0000i  -0.9127 - 1.4086i
  -0.3339 - 0.7458i  -0.9127 + 1.4086i   2.0000 + 0.0000i
```

The diagonal terms of the matrix represent the sum of the two signal powers.

Covariance Matrix for Two Independent Signals with 10 dB SNR

Create a spatial covariance matrix for a 3-element, half-wavelength-spaced uniform line array. Assume there are two incoming signals with unit power and there is additive noise with –10 dB power.

```
N = 3;
d = 0.5;
elementPos = (0:N-1)*d;
xcov = sensorcov(elementPos,[30 35],db2pow(-10))
```

```
xcov = 3×3 complex

   2.1000 + 0.0000i  -0.2291 - 1.9734i  -1.8950 + 0.4460i
  -0.2291 + 1.9734i   2.1000 + 0.0000i  -0.2291 - 1.9734i
  -1.8950 - 0.4460i  -0.2291 + 1.9734i   2.1000 + 0.0000i
```

The diagonal terms represent the two signal powers plus noise power at each sensor.

Covariance Matrix for Two Correlated Signals with 10 dB SNR

Compute the covariance matrix for a 3-element half-wavelength spaced line array when there is some correlation between two signals. The correlation can model, for example, multipath propagation caused by reflection from a surface. Assume an additive noise power value of –10 dB.

```
N = 3;
d = 0.5;
elementPos = (0:N-1)*d;
scov = [1, 0.8; 0.8, 1];
xcov = sensorcov(elementPos,[30 35],db2pow(-10),scov)
```

```
xcov = 3×3 complex

   3.7000 + 0.0000i  -0.4124 - 3.5521i  -3.4111 + 0.8028i
  -0.4124 + 3.5521i   3.6574 + 0.0000i  -0.4026 - 3.4682i
  -3.4111 - 0.8028i  -0.4026 + 3.4682i   3.5321 + 0.0000i
```

Input Arguments

pos — Positions of array sensor elements
1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array, specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero. When `pos` is a 2-by-*N* matrix, it represents the *(y,z)*-coordinates of the sensor elements of a planar array. This array is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. When `pos` is a 3-by-*N* matrix, then the array can have an arbitrary shape. Sensor positions are in terms of signal wavelength.

Example: `[0,0,0; 0.1,0.4,0.3; 1,1,1]`

Data Types: `double`

ang — Arrival directions of incoming signals
1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Arrival directions of incoming signals specified as a 1-by-*M* vector or a 2-by-*M* matrix, where *M* is the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the incoming signal `[az;el]`. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the arrival direction vector onto the *xy* plane. It is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the arrival direction vector and *xy*-plane. It is positive when measured towards the *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero. Angle units are specified in degrees.

Example: [45;0]

Data Types: `double`

ncov — Noise spatial covariance matrix
0 (default) | non-negative real-valued scalar | *1*-by-*N* non-negative real-valued vector | *N*-by-*N* positive definite, complex-valued matrix

Noise spatial covariance matrix specified as a non-negative, real-valued scalar, a non-negative, 1-by-*N* real-valued vector or an *N*-by-*N*, positive definite, complex-valued matrix. In this argument, *N* is the number of sensor elements. Using a non-negative scalar results in a noise spatial covariance matrix that has identical white noise power values (in watts) along its diagonal and has off-diagonal values of zero. Using a non-negative real-valued vector results in a noise spatial covariance that has diagonal values corresponding to the entries in `ncov` and has off-diagonal entries of zero. The diagonal entries represent the independent white noise power values (in watts) in each sensor. If `ncov` is *N*-by-*N* matrix, this value represents the full noise spatial covariance matrix between all sensor elements.

Example: [1,1,4,6]

Data Types: `double`
Complex Number Support: Yes

scov — Signal covariance matrix
1 (default) | non-negative real-valued scalar | *1*-by-*M* non-negative real-valued vector | *N*-by-*M* positive semidefinite, complex-valued matrix

Signal covariance matrix specified as a non-negative, real-valued scalar, a *1*-by-*M* non-negative, real-valued vector or an *M*-by-*M* positive semidefinite, matrix representing the covariance matrix between *M* signals. The number of signals is specified in `ang`. If `scov` is a nonnegative scalar, it assigns the same power (in watts) to all incoming signals which are assumed to be uncorrelated. If `scov` is a 1-by-*M* vector, it assigns the separate power values (in watts) to each incoming signal which are also assumed to be uncorrelated. If `scov` is an *M*-by-*M* matrix, then it represents the full covariance matrix between all incoming signals.

Example: [1 0 ; 0 2]

Data Types: `double`
Complex Number Support: Yes

Output Arguments

xcov — Sensor spatial covariance matrix
complex-valued N-by-N matrix

Sensor spatial covariance matrix returned as a complex-valued, N-by-N matrix. In this matrix, N represents the number of sensor elements of the array.

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York, NY: Wiley-Interscience, 2002.

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`cbfweights` | `lcmvweights` | `mvdrweights` | `steervec` | `sensorsig` | `phased.SteeringVector`

sensorsig

Simulate received signal at sensor array

Syntax

```
x = sensorsig(pos,ns,ang)
x = sensorsig(pos,ns,ang,ncov)
x = sensorsig(pos,ns,ang,ncov,scov)
x = sensorsig(pos,ns,ang,ncov,scov,'Taper',taper)
[x,rt] = sensorsig( ___ )
[x,rt,r] = sensorsig( ___ )
```

Description

`x = sensorsig(pos,ns,ang)` simulates the received narrowband plane wave signals at a sensor array. `pos` represents the positions of the array elements, each of which is assumed to be isotropic. `ns` indicates the number of snapshots of the simulated signal. `ang` represents the incoming directions of each plane wave signal. The plane wave signals are assumed to be constant-modulus signals with random phases.

`x = sensorsig(pos,ns,ang,ncov)` describes the noise across all sensor elements. `ncov` specifies the noise power or covariance matrix. The noise is a Gaussian distributed signal.

`x = sensorsig(pos,ns,ang,ncov,scov)` specifies the power or covariance matrix for the incoming signals.

`x = sensorsig(pos,ns,ang,ncov,scov,'Taper',taper)` specifies the array taper as a comma-separated pair consisting of `'Taper'` and a scalar or column vector.

`[x,rt] = sensorsig( ___ )` also returns the theoretical covariance matrix of the received signal, using any of the input arguments in the previous syntaxes.

`[x,rt,r] = sensorsig( ___ )` also returns the sample covariance matrix of the received signal.

Examples

Received Signal and Direction-of-Arrival Estimation

Simulate the received signal at an array, and use the data to estimate the arrival directions.

Create an 8-element uniform linear array whose elements are spaced half a wavelength apart.

```
fc = 3e8;
c = 3e8;
lambda = c/fc;
array = phased.ULA(8,lambda/2);
```

Simulate 100 snapshots of the received signal at the array. Assume there are two signals, coming from azimuth 30° and 60°, respectively. The noise is white across all array elements, and the SNR is 10 dB.

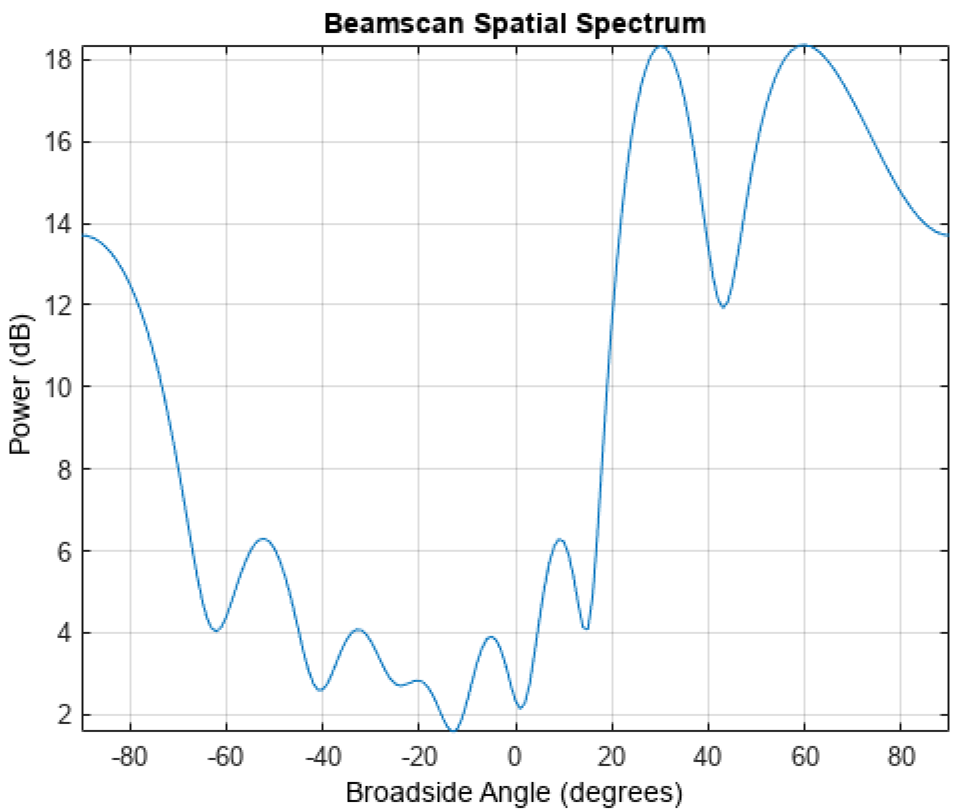
```
x = sensorsig(getElementPosition(array)/lambda,...
   100,[30 60],db2pow(-10));
```

Use a beamscan spatial spectrum estimator to estimate the arrival directions, based on the simulated data.

```
estimator = phased.BeamscanEstimator('SensorArray',array,...
   'PropagationSpeed',c,'OperatingFrequency',fc,...
   'DOAOutputPort',true,'NumSignals',2);
[~,ang_est] = estimator(x);
```

Plot the spatial spectrum resulting from the estimation process.

```
plotSpectrum(estimator)
```



The plot shows peaks at 30° and 60°.

Signals With Different Power Levels

Simulate receiving two uncorrelated incoming signals that have different power levels. A vector named `scov` stores the power levels.

Create an 8-element uniform linear array whose elements are spaced half a wavelength apart.

```
fc = 3e8;
c = 3e8;
```

```
lambda = c/fc;
ha = phased.ULA(8,lambda/2);
```

Simulate 100 snapshots of the received signal at the array. Assume that one incoming signal originates from 30 degrees azimuth and has a power of 3 W. A second incoming signal originates from 60 degrees azimuth and has a power of 1 W. The two signals are not correlated with each other. The noise is white across all array elements, and the SNR is 10 dB.

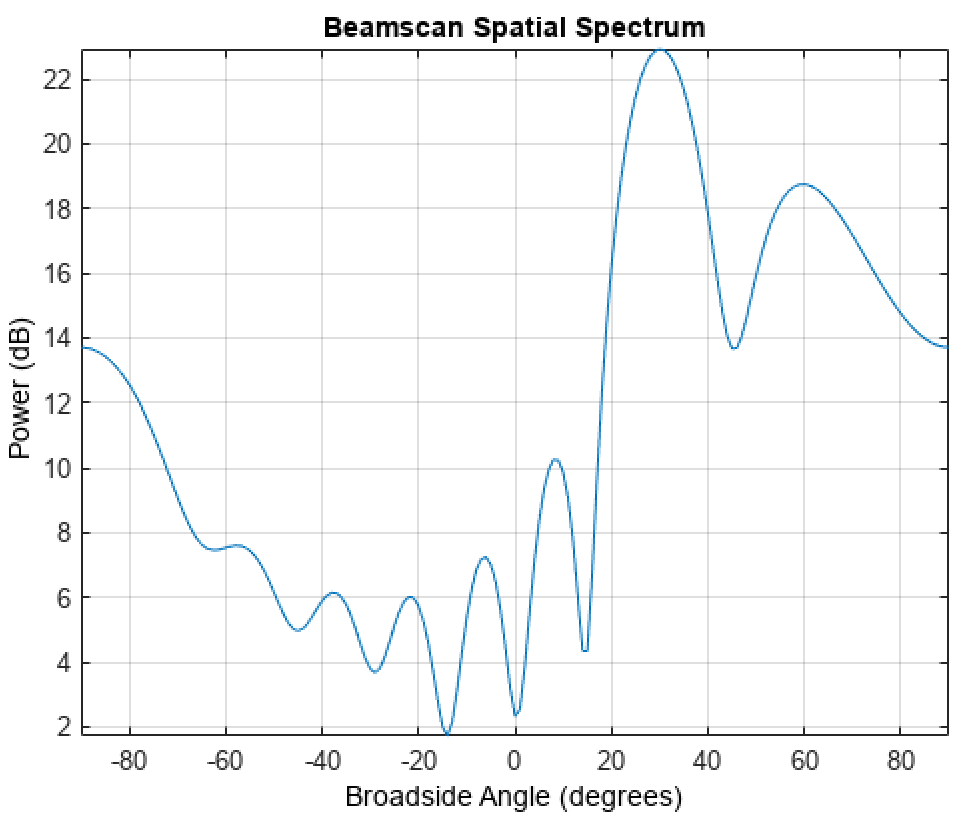
```
ang = [30 60];
scov = [3 1];
x = sensorsig(getElementPosition(ha)/lambda,...
   100,ang,db2pow(-10),scov);
```

Use a beamscan spatial spectrum estimator to estimate the arrival directions, based on the simulated data.

```
hdoa = phased.BeamscanEstimator('SensorArray',ha,...
   'PropagationSpeed',c,'OperatingFrequency',fc,...
   'DOAOutputPort',true,'NumSignals',2);
[~,ang_est] = step(hdoa,x);
```

Plot the spatial spectrum resulting from the estimation process.

```
plotSpectrum(hdoa);
```



The plot shows a high peak at 30 degrees and a lower peak at 60 degrees.

Reception of Correlated Signals

Simulate the reception of three signals, two of which are correlated.

Create a signal covariance matrix in which the first and third of three signals are correlated with each other.

```
scov = [1 0 0.6;...
        0 2 0;...
        0.6 0 1];
```

Simulate receiving 100 snapshots of three incoming signals from 30°, 40°, and 60° azimuth, respectively. The array that receives the signals is an 8-element uniform linear array whose elements are spaced one-half wavelength apart. The noise is white across all array elements, and the SNR is 10 dB.

```
pos = (0:7)*0.5;
ns = 100;
ang = [30 40 60];
ncov = db2pow(-10);
x = sensorsig(pos,ns,ang,ncov,scov);
```

Theoretical and Empirical Covariance of Received Signal

Simulate receiving a signal at a URA. Compare the signal theoretical covariance with its sample covariance.

Create a 2-by-2 uniform rectangular array having elements spaced 1/4-wavelength apart.

```
pos = 0.25 * [0 0 0 0; -1 1 -1 1; -1 -1 1 1];
```

Define the noise power independently for each of the four array elements. Each entry in `ncov` is the noise power of an array element. This element position is the corresponding column in `pos`. Assume the noise is uncorrelated across elements.

```
ncov = db2pow([-9 -10 -10 -11]);
```

Simulate 100 snapshots of the received signal at the array, and store the theoretical and empirical covariance matrices. Assume that one incoming signal originates from 30° azimuth and 10° elevation. A second incoming signal originates from 50° azimuth and 0° elevation. The signals have a power of 1 W and are uncorrelated.

```
ns = 100;
ang1 = [30; 10];
ang2 = [50; 0];
ang = [ang1, ang2];
rng default
[x,rt,r] = sensorsig(pos,ns,ang,ncov);
```

View the magnitudes of the theoretical covariance and sample covariance.

```
abs(rt)
```

```
ans = 4×4

    2.1259    1.8181    1.9261    1.9754
    1.8181    2.1000    1.5263    1.9261
    1.9261    1.5263    2.1000    1.8181
    1.9754    1.9261    1.8181    2.0794
```

```
abs(r)
```

```
ans = 4×4

    2.2107    1.7961    2.0205    1.9813
    1.7961    1.9858    1.5163    1.8384
    2.0205    1.5163    2.1762    1.8072
    1.9813    1.8384    1.8072    2.0000
```

Correlation of Noise Between Sensors

Simulate receiving a signal at a ULA, where the noise between different sensors is correlated.

Create a 4-element uniform linear array whose elements are spaced one-half wavelength apart.

```
pos = 0.5 * (0:3);
```

Define the noise covariance matrix. The value in the (*k*,_j_) position in the `ncov` matrix is the covariance between the *k* and *j* array elements listed in array.

```
ncov = 0.1 * [1 0.1 0 0; 0.1 1 0.1 0; 0 0.1 1 0.1; 0 0 0.1 1];
```

Simulate 100 snapshots of the received signal at the array. Assume that one incoming signal originates from 60° azimuth.

```
ns = 100;
ang = 60;
[x,rt,r] = sensorsig(pos,ns,ang,ncov);
```

View the theoretical and sample covariance matrices for the received signal.

```
rt,r
```

```
rt = 4×4 complex

   1.1000 + 0.0000i  -0.9027 - 0.4086i   0.6661 + 0.7458i  -0.3033 - 0.9529i
  -0.9027 + 0.4086i   1.1000 + 0.0000i  -0.9027 - 0.4086i   0.6661 + 0.7458i
   0.6661 - 0.7458i  -0.9027 + 0.4086i   1.1000 + 0.0000i  -0.9027 - 0.4086i
  -0.3033 + 0.9529i   0.6661 - 0.7458i  -0.9027 + 0.4086i   1.1000 + 0.0000i
```

```
r = 4×4 complex

   1.1059 + 0.0000i  -0.8681 - 0.4116i   0.6550 + 0.7017i  -0.3151 - 0.9363i
  -0.8681 + 0.4116i   1.0037 + 0.0000i  -0.8458 - 0.3456i   0.6578 + 0.6750i
   0.6550 - 0.7017i  -0.8458 + 0.3456i   1.0260 + 0.0000i  -0.8775 - 0.3753i
  -0.3151 + 0.9363i   0.6578 - 0.6750i  -0.8775 + 0.3753i   1.0606 + 0.0000i
```

Input Arguments

pos — Positions of elements in sensor array
1-by-N vector | 2-by-N matrix | 3-by-N matrix

Positions of elements in sensor array, specified as an N-column vector or matrix. The values in the matrix are in units of signal wavelength. For example, `[0 1 2]` describes three elements that are spaced one signal wavelength apart. N is the number of elements in the array.

Dimensions of `pos`:

- For a linear array along the y axis, specify the y coordinates of the elements in a 1-by-N vector.
- For a planar array in the yz plane, specify the y and z coordinates of the elements in columns of a 2-by-N matrix.
- For an array of arbitrary shape, specify the x, y, and z coordinates of the elements in columns of a 3-by-N matrix.

Data Types: `double`

ns — Number of snapshots of simulated signal
positive integer scalar

Number of snapshots of simulated signal, specified as a positive integer scalar. The function returns this number of samples per array element.

Data Types: `double`

ang — Directions of incoming plane wave signals
1-by-M vector | 2-by-M matrix

Directions of incoming plane wave signals, specified as an M-column vector or matrix in degrees. M is the number of incoming signals.

Dimensions of `ang`:

- If `ang` is a 2-by-M matrix, each column specifies a direction. Each column is in the form `[azimuth; elevation]`. The azimuth angle on page 2-425 must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.
- If `ang` is a 1-by-M vector, each entry specifies an azimuth angle. In this case, the corresponding elevation angle is assumed to be 0.

Data Types: `double`

ncov — Noise characteristics
0 (default) | nonnegative scalar | 1-by-N vector of positive numbers | N-by-N positive definite matrix

Noise characteristics, specified as a nonnegative scalar, 1-by-N vector of positive numbers, or N-by-N positive definite matrix.

Dimensions of `ncov`:

- If `ncov` is a scalar, it represents the noise power of the white noise across all receiving sensor elements, in watts. In particular, a value of `0` indicates that there is no noise.

- If `ncov` is a 1-by-N vector, each entry represents the noise power of one of the sensor elements, in watts. The noise is uncorrelated across sensors.
- If `ncov` is an N-by-N matrix, it represents the covariance matrix for the noise across all sensor elements.

Data Types: `double`

`scov` — Incoming signal characteristics
1 (default) | positive scalar | 1-by-M vector of positive numbers | M-by-M positive semidefinite matrix

Incoming signal characteristics, specified as a positive scalar, 1-by-M vector of positive numbers, or M-by-M positive semidefinite matrix.

Dimensions of `scov`:

- If `scov` is a scalar, it represents the power of all incoming signals, in watts. In this case, all incoming signals are uncorrelated and share the same power level.
- If `scov` is a 1-by-M vector, each entry represents the power of one of the incoming signals, in watts. In this case, all incoming signals are uncorrelated with each other.
- If `scov` is an M-by-M matrix, it represents the covariance matrix for all incoming signals. The matrix describes the correlation among the incoming signals. In this case, `scov` can be real or complex.

Data Types: `double`

`taper` — Array element taper
1 (default) | scalar | *N*-by-1 column vector

Array element taper, specified as a scalar or complex-valued *N*-by-1 column vector. The dimension *N* is the number of array elements. If `taper` is a scalar, all elements in the array use the same value. If `taper` is a vector, each entry specifies the taper applied to the corresponding array element.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

`x` — Received signal
complex `ns`-by-N matrix

Received signal at sensor array, returned as a complex `ns`-by-N matrix. Each column represents the received signal at the corresponding element of the array. Each row represents a snapshot.

`rt` — Theoretical covariance matrix
complex N-by-N matrix

Theoretical covariance matrix of the received signal, returned as a complex N-by-N matrix.

`r` — Sample covariance matrix
complex N-by-N matrix

Sample covariance matrix of the received signal, returned as a complex N-by-N matrix. N is the number of array elements. The function derives this matrix from `x`.

Note If you specify this output argument, consider making `ns` greater than or equal to N. Otherwise, `r` is rank deficient.

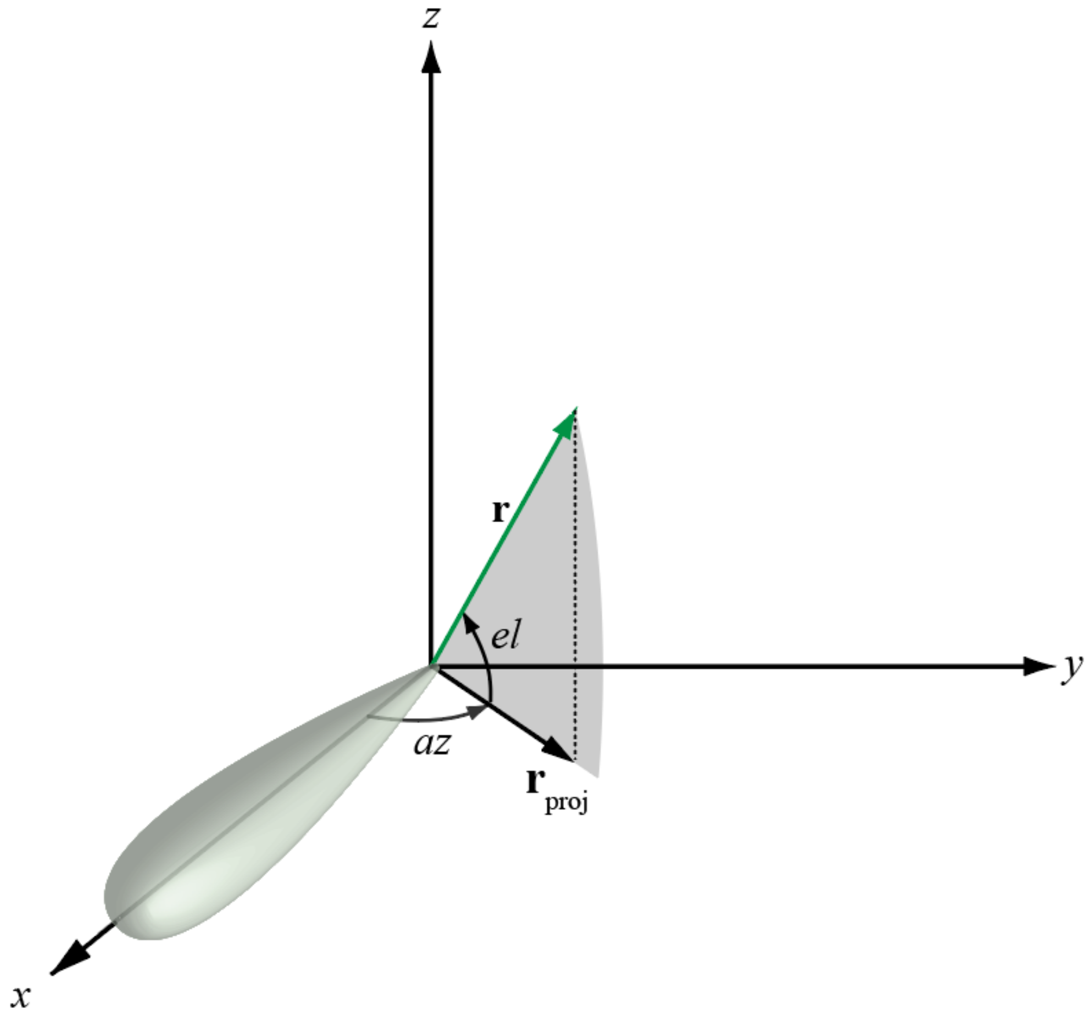
More About

Azimuth Angle, Elevation Angle

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



Version History

Introduced in R2012b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phased.SteeringVector`

shnidman

Required SNR using Shnidman’s equation

Syntax

```
SNR = shnidman(Pd,Pfa)
SNR = shnidman(Pd,Pfa,N)
SNR = shnidman(Pd,Pfa,N,Sw)
```

Description

`SNR = shnidman(Pd,Pfa)` returns the required signal-to-noise ratio in decibels for the specified detection and false-alarm probabilities using Shnidman's equation. The SNR is determined for a single pulse and a “Swerling Case Number” on page 2-429 of 0, a nonfluctuating target.

`SNR = shnidman(Pd,Pfa,N)` returns the required SNR for a nonfluctuating target based on the noncoherent integration of `N` pulses.

`SNR = shnidman(Pd,Pfa,N,Sw)` returns the required SNR for the Swerling case number `Sw`.

Examples

Compute Single-Pulse SNR

Find and compare the required single-pulse SNR for Swerling cases I and III. The Swerling case I has no dominant scatterer while the Swerling case III has a dominant scatterer.

Specify the false-alarm and detection probabilities.

```
pfa = 1e-6:1e-5:.001;
Pd = 0.9;
```

Allocate arrays for plotting.

```
SNR_Sw1 = zeros(1,length(pfa));
SNR_Sw3 = zeros(1,length(pfa));
```

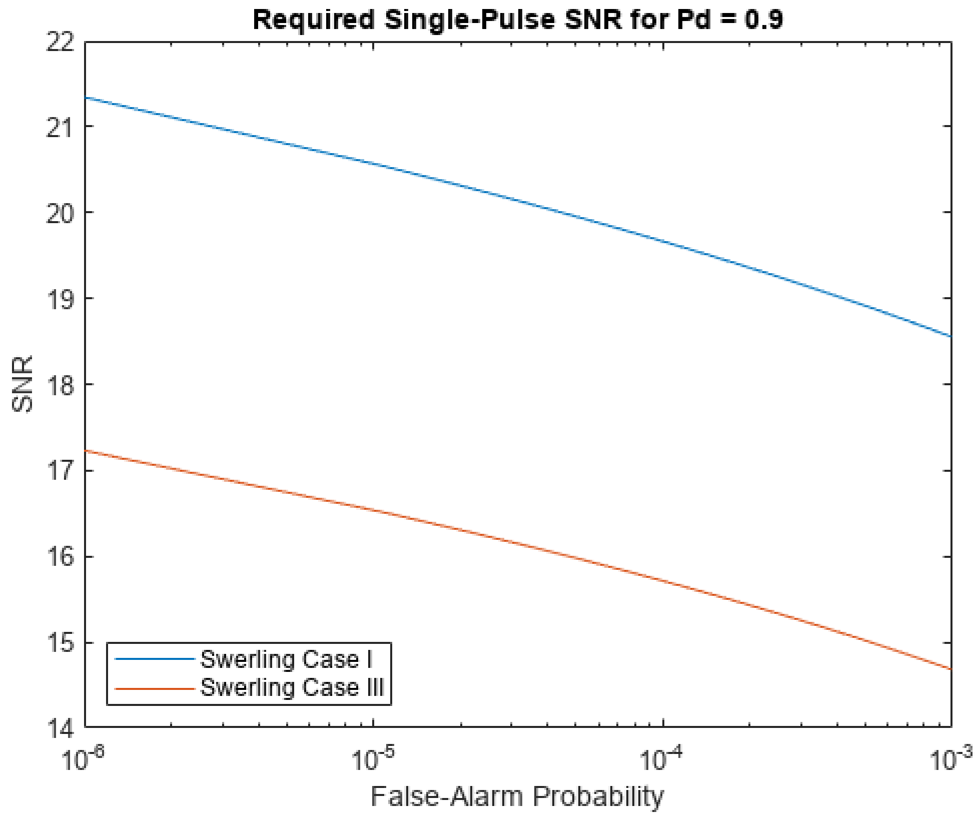
Loop over PFAs for both scatterer cases.

```
for j=1:length(pfa)
    SNR_Sw1(j) = shnidman(Pd,pfa(j),1,1);
    SNR_Sw3(j) = shnidman(Pd,pfa(j),1,3);
end
```

Plot the SNR vs PFA.

```
semilogx(pfa,SNR_Sw1)
hold on
semilogx(pfa,SNR_Sw3)
hold off
```

```
xlabel("False-Alarm Probability")
ylabel("SNR")
title("Required Single-Pulse SNR for Pd = "+Pd)
legend("Swerling Case "+["I" "III"],Location="southwest")
```



The presence of a dominant scatterer reduces the required SNR for the specified detection and false-alarm probabilities.

Input Arguments

Pd — Probability of detection
positive scalar

Probability of detection, specified as a positive scalar.

Data Types: double

Pfa — Probability of false alarm
positive scalar

Probability of false alarm, specified as a positive scalar.

Data Types: double

N — Number of pulses for noncoherent integration
1 (default) | positive scalar

Number of pulses for noncoherent integration, specified as a positive scalar.

Data Types: `double`

`Sw` — Swerling case number
`0` (default) | `1` | `2` | `3` | `4`

Swerling case number, specified as `0`, `1`, `2`, `3`, or `4`. For more information, see “Swerling Case Number” on page 2-429

Data Types: `double`

More About

Shnidman's Equation

Shnidman's equation is a series of equations that yield an estimate of the SNR required for a specified false-alarm and detection probability. Like Albersheim's equation, Shnidman's equation is applicable to a single pulse or the noncoherent integration of N pulses. Unlike Albersheim's equation, Shnidman's equation holds for square-law detectors and is applicable to fluctuating targets. An important parameter in Shnidman's equation is the Swerling case number.

Swerling Case Number

The Swerling case numbers characterize the detection problem for fluctuating pulses in terms of:

- A decorrelation model for the received pulses
- The distribution of scatterers affecting the probability density function (PDF) of the target radar cross section (RCS).

The Swerling case numbers consider all combinations of two decorrelation models (scan-to-scan; pulse-to-pulse) and two RCS PDFs (based on the presence or absence of a dominant scatterer).

Swerling Case Number	Description
0 (alternatively designated as 5)	Nonfluctuating pulses.
I	Scan-to-scan decorrelation. Rayleigh/exponential PDF–A number of randomly distributed scatterers with no dominant scatterer.
II	Pulse-to-pulse decorrelation. Rayleigh/ exponential PDF– A number of randomly distributed scatterers with no dominant scatterer.
III	Scan-to-scan decorrelation. Chi-square PDF with 4 degrees of freedom. A number of scatterers with one dominant.
IV	Pulse-to-pulse decorrelation. Chi-square PDF with 4 degrees of freedom. A number of scatterers with one dominant.

Version History
Introduced in R2011a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

See Also
`albersheim`

sonareqsl

Compute source level using the sonar equation

Syntax

```
SL = sonareqsl(SNR,NL,DI,TL)
SL = sonareqsl(SNR,NL,DI,TL,TS)
```

Description

`SL = sonareqsl(SNR,NL,DI,TL)` returns the source level of a signal, `SL`, required to achieve a specified received signal-to-noise ratio, `SNR`. Source level is computed using the “Sonar Equation”. Specify the received noise level, `NL`, receiver directivity index, `DI`, and the transmission loss, `TL`. Use this syntax to evaluate passive sonar system performance.

`SL = sonareqsl(SNR,NL,DI,TL,TS)` returns the source level taking into account the target strength `TS`. Use this syntax to evaluate active sonar system performance, where the transmitted signal is reflected from a target. `TL` represents one-way transmission loss.

Examples

Estimate Source Level from Passive Sonar Equation

Estimate the source level of a signal arriving from a source with an SNR of 10 dB. The noise level is 75 dB, the receive array directivity index is 25 dB, and the transmission loss is 140 dB.

```
SNR = 10;
NL = 75.0;
DI = 25.0;
TL = 140.0;
SL = sonareqsl(SNR,NL,DI,TL)
```

```
SL = 200
```

Estimate Source Level from Active Sonar Equation

Estimate the source level of a signal transmitted by a source with SNR of 15 dB and reflected from a target with 25 dB//1 m^2 target strength. The noise level is 45 dB//1 µPa, the receive array directivity index is 25 dB, and the one-way transmission loss is 60 dB.

```
SNR = 15.0;
NL = 45.0;
DI = 25.0;
TL = 60.0;
TS = 25.0;
SL = sonareqsl(SNR,NL,DI,TL,TS)
```

```
SL = 130
```

Input Arguments

SNR — Received signal-to-noise ratio
scalar

Received signal-to-noise ratio, specified as a scalar. Units are in dB.

Example: 10

Data Types: double

NL — Received noise level
scalar

Received noise level, specified as a scalar. Noise level is the ratio of the noise intensity to a reference intensity, converted to dB. The reference intensity is the intensity of a sound wave having a root-mean-square (rms) pressure of 1 μPa. Units are in dB//1 μPa.

Example: 70

Data Types: double

DI — Receiver directivity index
scalar

Receiver directivity index, specified as a scalar. Units are in dB.

Example: 30

Data Types: double

TL — Transmission loss
positive scalar

Transmission loss (*TL*), specified as a positive scalar. Transmission loss is the attenuation of sound intensity as the sound propagates through the underwater channel. Transmission loss is defined as the ratio of sound intensity at 1 m from a source to the sound intensity at distance *R*. For active sonar, TL represents one-way transmission loss.

$$TL = 10\log\frac{I_s}{I(R)}$$

Units are in dB.

Example: 120

Data Types: double

TS — Target strength
scalar

Target strength, specified as a scalar. Target strength is the ratio of the intensity of a reflected signal at 1 m from a target to the incident intensity. Target strength is the sonar analog to radar cross section. Units are in dB//1 m^2.

Example: 5

Data Types: double

Output Arguments

SL — Sonar source level
scalar

Sonar source level, returned as a scalar. Source level is the ratio of the source intensity to a reference intensity, converted to dB. The reference intensity is the intensity of a sound wave having an rms pressure of 1 µPa. Units are in dB//1 µPa.

Version History

Introduced in R2017b

References

[1] Ainslie M. A. and J.G. McColm. "A simplified formula for viscous and chemical absorption in sea water." *Journal of the Acoustical Society of America*, Vol. 103, Number 3, 1998, pp. 1671--1672.

[2] Urick, Robert J, *Principles of Underwater Sound*, 3rd ed. Peninsula Publishing, Los Altos, CA, 1983.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`range2tl` | `sonareqsnr` | `sonareqtl` | `tl2range`

Topics
“Sonar Equation”
“Element Directivity”

sonareqsnr

Compute SNR using the sonar equation

Syntax

```
SNR = sonareqsnr(SL,NL,DI,TL)
SNR = sonareqsnr(SL,NL,DI,TL,TS)
```

Description

`SNR = sonareqsnr(SL,NL,DI,TL)` returns the received signal-to-noise ratio, `SNR`, from the source level, `SL`, received noise level, `NL`, receiver directivity index, `DI`, and transmission loss, `TL`. SNR is computed using the “Sonar Equation”. Use this syntax to evaluate passive sonar system performance.

`SNR = sonareqsnr(SL,NL,DI,TL,TS)` returns SNR taking into account the target strength `TS`. Use this syntax to evaluate active sonar system performance, where the transmitted signal is reflected from a target.

Examples

Estimate SNR from Passive Sonar Equation

Estimate the SNR of a signal arriving from a source with a source level of 200 dB. The noise level is 75 dB, the receive array directivity index is 25 dB, and the transmission loss is 140 dB.

```
SL = 200.0;
NL = 75.0;
DI = 25.0;
TL = 140.0;
SNR = sonareqsnr(SL,NL,DI,TL)
```

```
SNR = 10
```

Estimate SNR from Active Sonar Equation

Estimate the SNR of a signal transmitted by a source with a source level of 130 dB//1 µPa and reflected from a target with 25 dB//1 m^2 target strength. The noise level is 45 dB//1 µPa, the receive array directivity is 25 dB, and the one-way transmission loss is 60 dB.

```
SL = 130.0;
NL = 45.0;
DI = 25.0;
TL = 60.0;
TS = 25.0;
SNR = sonareqsnr(SL,NL,DI,TL,TS)
```

```
SNR = 15
```

Input Arguments

SL — Sonar source level

scalar

Sonar source level, specified as a scalar. Source level is the ratio of the source intensity to a reference intensity, converted to dB. The reference intensity is the intensity of a sound wave having a root-mean-square (rms) pressure of 1 μPa. Units are in dB//1 μPa.

Example: `90`

Data Types: `double`

NL — Received noise level

scalar

Received noise level, specified as a scalar. Noise level is the ratio of the noise intensity to a reference intensity, converted to dB. The reference intensity is the intensity of a sound wave having a root-mean-square (rms) pressure of 1 μPa. Units are in dB//1 μPa.

Example: `70`

Data Types: `double`

DI — Receiver directivity index

scalar

Receiver directivity index, specified as a scalar. Units are in dB.

Example: `30`

Data Types: `double`

TL — Transmission loss

positive scalar

Transmission loss (*TL*), specified as a positive scalar. Transmission loss is the attenuation of sound intensity as the sound propagates through the underwater channel. Transmission loss is defined as the ratio of sound intensity at 1 m from a source to the sound intensity at distance *R*. For active sonar, `TL` represents one-way transmission loss.

$$TL = 10\log\frac{I_s}{I(R)}$$

Units are in dB.

Example: `120`

Data Types: `double`

TS — Target strength

scalar

Target strength, specified as a scalar. Target strength is the ratio of the intensity of a reflected signal at 1 m from a target to the incident intensity. Target strength is the sonar analog to radar cross section. Units are in dB//1 m^2.

Example: `5`

Data Types: `double`

Output Arguments

SNR — Received signal-to-noise ratio
real scalar

Received signal-to-noise ratio, returned as a scalar.

Data Types: `double`

Version History

Introduced in R2017b

References

[1] Ainslie M. A. and J.G. McColm. "A simplified formula for viscous and chemical absorption in sea water." *Journal of the Acoustical Society of America*, Vol. 103, Number 3, 1998, pp. 1671--1672.

[2] Urick, Robert J, *Principles of Underwater Sound*, 3rd ed. Peninsula Publishing, Los Altos, CA, 1983.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`range2tl` | `sonareqsl` | `sonareqtl` | `tl2range`

Topics
"Sonar Equation"
"Element Directivity"

sonareqtl

Compute transmission loss using the sonar equation

Syntax

```
TL = sonareqtl(SL,SNR,NL,DI)
TL = sonareqtl(SL,SNR,NL,DI,TS)
```

Description

`TL = sonareqtl(SL,SNR,NL,DI)` returns the transmission loss of a signal from source to receiver that produces the signal-to-noise ratio, `SNR`. Transmission loss is computed using the "Sonar Equation". Required inputs are the source level, `SL`, received noise level, `NL`, and receiver directivity index, `DI`. Use this syntax to evaluate passive sonar system performance.

`TL = sonareqtl(SL,SNR,NL,DI,TS)` returns the one-way transmission loss. The signal is reflected from a target with a target strength, `TS`. Use this syntax to evaluate active sonar system performance, where the transmitted signal is reflected from a target.

Examples

Estimate Transmission Loss from Passive Sonar Equation

Estimate the transmission loss of a signal arriving from a source with source level of 200 dB. The received SNR is 10 dB, the noise level is 75 dB, and the receive array directivity index is 25 dB.

```
SNR = 10;
SL = 200.0;
NL = 75.0;
DI = 25.0;
TL = sonareqtl(SL,SNR,NL,DI)
```

```
TL = 140
```

Estimate Transmission Loss from Active Sonar Equation

Estimate the one-way transmission loss of a signal transmitted by a source with source level of 130 dB//1 µPa and reflected from a target with 25 dB//1 m^2 target strength. The noise level is 45 dB//1 µPa, the receive array directivity is 25 dB.

```
SL = 130.0;
SNR = 15.0;
NL = 45.0;
DI = 25.0;
TS = 25.0;
TL = sonareqtl(SL,SNR,NL,DI,TS)
```

```
TL = 60
```

Input Arguments

SL — Sonar source level

scalar

Sonar source level, specified as a scalar. Source level is the ratio of the source intensity to a reference intensity, converted to dB. The reference intensity is the intensity of a sound wave having a root-mean-square (rms) pressure of 1 µPa. Units are in dB//1 µPa.

Example: `90`

Data Types: `double`

SNR — Received signal-to-noise ratio

scalar

Received signal-to-noise ratio, specified as a scalar. Units are in dB.

Example: `10`

Data Types: `double`

NL — Received noise level

scalar

Received noise level, specified as a scalar. Noise level is the ratio of the noise intensity to a reference intensity, converted to dB. The reference intensity is the intensity of a sound wave having a root-mean-square (rms) pressure of 1 µPa. Units are in dB//1 µPa.

Example: `70`

Data Types: `double`

DI — Receiver directivity index

scalar

Receiver directivity index, specified as a scalar. Units are in dB.

Example: `30`

Data Types: `double`

TS — Target strength

scalar

Target strength, specified as a scalar. Target strength is the ratio of the intensity of a reflected signal at 1 m from a target to the incident intensity. Target strength is the sonar analog to radar cross section. Units are in dB//1 m^2.

Example: `5`

Data Types: `double`

Output Arguments

TL — Transmission loss

positive scalar

Transmission loss, returned as a positive scalar. Transmission loss is the attenuation of sound intensity as the sound propagates through the underwater channel. Transmission loss is defined as the ratio of sound intensity at 1 m from a source to the sound intensity at distance *R*. When target strength, TS, is specified, transmission loss is two-way.

Version History

Introduced in R2017b

References

[1] Ainslie M. A. and J.G. McColm. "A simplified formula for viscous and chemical absorption in sea water." *Journal of the Acoustical Society of America*, Vol. 103, Number 3, 1998, pp. 1671--1672.

[2] Urick, Robert J, *Principles of Underwater Sound*, 3rd ed. Peninsula Publishing, Los Altos, CA, 1983.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`range2tl` | `sonareqsl` | `tl2range` | `sonareqsnr`

Topics

"Sonar Equation"
"Element Directivity"

scatteringchanmtx

Scattering channel matrix

Syntax

```
chmat = scatteringchanmtx(txarraypos,rxarraypos,numscat)
chmat = scatteringchanmtx(txarraypos,rxarraypos,numscat,angrange)
chmat = scatteringchanmtx(txarraypos,rxarraypos,txang,rxang,G)
```

Description

`chmat = scatteringchanmtx(txarraypos,rxarraypos,numscat)` returns the channel matrix, `chmat`, for a MIMO channel consisting of a transmitting array, a receiver array, and multiple scatterers. The transmitting array is located at `txarraypos` and the receiving array at `rxarraypos`. `numscat` is the number of point scatterers.

The function generates `numscat` random transmission directions and `numscat` random receiving directions. The channel matrix describes multipath propagation through the `numscat` paths. By assumption, all paths arrive at the receiving array simultaneously implying that the channel is frequency flat. Flat frequency means that the spectrum of the signal is not changed. Path gains are derived from a zero-mean, unit-variance, complex-valued normal distribution.

`chmat = scatteringchanmtx(txarraypos,rxarraypos,numscat,angrange)` also specifies the angular range, `angrange`, for transmitting and receiving angles.

`chmat = scatteringchanmtx(txarraypos,rxarraypos,txang,rxang,G)` also specifies transmitting angles, `txang`, receiving angles, `rxang`, and path gains, `G`.

Examples

Compute Channel Matrix for Random Signal Paths

Compute the channel matrix for a 13-element transmitting array and a 15-element receiving array. Assume that there are 17 randomly located scatterers. The arrays are uniform linear arrays with 0.45-wavelength spacing. The receiving array is 300 wavelengths away from the transmitting array. Use the channel matrix to compute a propagated signal from the transmitting array to the receiving array.

Specify the arrays. Element spacing is in units of wavelength.

```
numtx = 13;
sp = 0.45;
txpos = (0:numtx-1)*sp;
numrx = 15;
rxpos = 300 + (0:numrx-1)*sp;
```

Specify the number of scatterers and create the channel matrix.

```
numscat = 17;
chmat = scatteringchanmtx(txpos,rxpos,numscat);
```

Create a signal consisting of zeros and ones. Then, propagate the signal from the transmitter to receiver.

```
x = randi(2,[100 numtx])-1;
y = x*chmat;
```

Compute Channel Matrix for Constrained Random Signal Paths

Compute the channel matrix for a 4-by-4 transmitting URA array and a 5-by-5 receiving URA array. Assume that 17 scatterers are randomly located within a specified angular range. The element spacing for both arrays is one-half wavelength. The receive array is 500 wavelengths away from the transmitting array along the *x*-axis. Use the channel matrix to compute a propagated signal from the transmitting array to the receiving array. Constrain the angular span for the transmitting and receiving directions.

Specify the 4-by-4 transmitting array. Element spacing is in units of wavelength.

```
Nt = 4;
sp = 0.5;
ygridtx = (0:Nt-1)*sp - (Nt-1)/2*sp;
zgridtx = (0:Nt-1)*sp - (Nt-1)/2*sp;
[ytx,ztx] = meshgrid(ygridtx,zgridtx);
txpos = [zeros(1,Nt*Nt);ytx(:).';ztx(:).'];
```

Specify the 5-by-5 receiving array. Element spacing is in units of wavelength.

```
Nr = 5;
sp = 0.5;
ygridrx = (0:Nr-1)*sp - (Nr-1)/2*sp;
zgridrx = (0:Nr-1)*sp - (Nr-1)/2*sp;
[yrx,zrx] = meshgrid(ygridrx,zgridrx);
rxpos = [500*ones(1,Nr*Nr);yrx(:).';zrx(:).'];
```

Set the angular limits for transmitting and receiving.

- The azimuth angle limits for the transmitter are −45° to +45°.
- The azimuth angle limits for the receiver are −75° to +50°.
- The elevation angle limits for the transmitter are −12° to +12°.
- The elevation angle limits for the receiver are −30° to +30°.

```
angrange = [-45 45 -75 50; -12 12 -30 30];
```

Specify the number of scatterers and create the channel matrix.

```
numscat = 6;
chmat = scatteringchanmtx(txpos,rxpos,numscat,angrange);
```

Create a 100-sample signal consisting of zeros and ones. Then, propagate the signal from the transmitting array to the receiving array.

```
x = randi(2,[100 Nt*Nt])-1;
y = x*chmat;
```

Compute Channel Matrix for Specified Signal Paths

Compute the channel matrix for a 4-by-4 transmitting URA array and a 5-by-5 receiving URA array. Assume there are 3 scatterers with known directions. The element spacings for both arrays is one-half wavelength. The receive array is 500 wavelengths away from the transmitting array along the *x*-axis. Use the channel matrix to compute a propagated signal from the transmitting array to the receiving array. Specify the transmitting and receiving directions. The number of directions determines the number of scatterers.

Specify the 4-by-4 transmitting array. Element spacing is in units of wavelength.

```
Nt = 4;
sp = 0.5;
ygridtx = (0:Nt-1)*sp - (Nt-1)/2*sp;
zgridtx = (0:Nt-1)*sp - (Nt-1)/2*sp;
[ytx,ztx] = meshgrid(ygridtx,zgridtx);
txpos = [zeros(1,Nt*Nt);ytx(:).';ztx(:).'];
```

Specify the 5-by-5 receiving array. Element spacing is in units of wavelength.

```
Nr = 5;
sp = 0.5;
ygridrx = (0:Nr-1)*sp - (Nr-1)/2*sp;
zgridrx = (0:Nr-1)*sp - (Nr-1)/2*sp;
[yrx,zrx] = meshgrid(ygridrx,zgridrx);
rxpos = [500*ones(1,Nr*Nr);yrx(:).';zrx(:).'];
```

Specify the transmitting and receiving angles and the gains. Then, create the channel matrix.

```
txang = [20 -10 40; 0 12 -12];
rxang = [70 -5.5 27.2; 4 1 -10];
gains = [1 1+1i 2-3*1i];
chmat = scatteringchanmtx(txpos,rxpos,txang,rxang,gains);
```

Create a 100-sample signal consisting of zeros and ones. Then, propagate the signal from the transmitting array to the receiving array.

```
x = randi(2,[100 Nt*Nt])-1;
y = x*chmat;
```

Input Arguments

`txarraypos` — Positions of elements in transmitting array

real-valued 1-by-N_t row vector | real-valued 2-by-N_t matrix | real-valued 3-by-N_t matrix

Transmitting array element positions, specified as a real-valued 1-by-N_t row vector, 2-by-N_t matrix, or 3-by-N_t matrix. N_t is the number of elements in the transmitting array.

`txarraypos`	Dimensions of Transmitting Array
1-by-N_t row vector	All transmitting array elements lie along the *y*-axis. The vector specifies the *y*-coordinates of the array elements.

`txarraypos`	Dimensions of Transmitting Array
2-by-N_t matrix	All transmitting array elements lie in the *yz*-plane. Each column of the matrix specifies the *y* and *z* coordinates of an array element.
3-by-N_t matrix	The transmitting array elements have arbitrary 3-D coordinates. Each column of the matrix specifies the *x*, *y*, and *z* coordinates of an array element.

Units are in wavelengths.

Example: `[-2.0,-1.0,0.0,1.0,2.0]`

Data Types: `double`

`rxarraypos` — Positions of elements in receiving array
real-valued 1-by-N_r row vector | real-valued 2-by-N_r matrix | real-valued 3-by-N_r matrix

Receiving array element positions, specified as a real-valued 1-by-N_r row vector, 2-by-vmatrix, or 3-by-N_r matrix. N_t is the number of elements in the transmitting array.

`rxarraypos`	Dimensions of Receiving Array
1-by-N_r row vector	All receiving array elements lie along the *y*-axis. The vector specifies the *y*-coordinates of the array elements.
2-by-N_r matrix	All receiving array elements lie in the *yz*-plane. Each column of the matrix specifies the *y* and *z* coordinates of an array element.
3-by-N_r matrix	The receiving array elements have arbitrary 3-D coordinates. Each column of the matrix specifies the *x*, *y*, and *z* coordinates of an array element.

Units are in wavelengths.

Example: `[-2.0,-1.0,0.0,1.0,2.0]`

Data Types: `double`

`numscat` — Number of scatterers
positive integer

Number of scatters, specified as a positive integer

Example: 7

Data Types: `double`

`angrange` — Angular range of transmission and reception directions
real-valued 1-by-2 row vector | real-valued row 1-by-4 vector | real-valued 2-by-2 matrix | real-valued 2-by-4 matrix

Angular range of transmitting and receiving directions, specified as one of the values in this table.

Size of angrange	Angular range
real-valued 1-by-2 row vector	Specify the same azimuth angle direction span for the transmitting and receiving arrays by using the minimum and maximum azimuth angles, `[az_min az_max]`. The elevation direction span is –90° to +90°.
real-valued 1-by-4 row vector	Specify the azimuth angle direction range for the transmitting and receiving arrays by using `[tx_az_min tx_az_max rx_az_min rx_az_max]`. The first two values are the minimum and maximum of the transmitting array directions. The last two values are the minimum and maximum of the receiving array directions. The range of the elevation angles is –90° to +90°.
real-valued 2-by-2 matrix	Specify the same azimuth and elevation angle direction spans for the transmitting and receiving arrays by using the minimum and maximum azimuth and elevation angles, `[az_min az_max; el_min el_max]`.
real-valued 2-by-4 matrix	Specify separate azimuth and elevation angle direction spans by using `[tx_az_min tx_az_max rx_az_min rx_az_max; tx_el_min tx_el_max rx_el_min rx_el_max]`. The first and second columns form the transmitting array direction span. The last two columns form the receiving array direction span.

Units are in degrees.

Example: `[-45 45 -30 30; -10 20 -5 30]`

Data Types: `double`

`txang` — Transmission path angles

real-valued 1-by-N_s row vector | real-valued 2-by-N_s matrix

Transmission path angles, specified as a real-valued 1-by-N_s row vector or a 2-by-N_s matrix. N_s is the number of scatterers specified by `numscat`.

- When `txang` is a vector, each element specifies the azimuth angle of a path. The elevation angle of the path is zero degrees.
- When `txang` is a matrix, each column specifies the azimuth and elevation angles of a path in the form `[az;el]`.

Example: `[4 -2; 0 35]`

Data Types: `double`

`rxang` — Receiving path angles

real-valued 1-by-N_s row vector | real-valued 2-by-N_s matrix

Receiving path angles, specified as a real-valued 1-by-N_s row vector or a 2-by-N_s matrix. N_s is the number of scatterers specified by `numscat`.

- When `rxang` is a vector, each element specifies the azimuth angle of a path. The elevation angle of the path is zero degrees.
- When `rxang` is a matrix, each column specifies the azimuth and elevation angles of a path in the form `[az;el]`.

Example: `[4 -2; 0 35]`

Data Types: `double`

`G` — Path gains
1-by-N_s complex-valued row vector

Path gains, specified as a 1-by-N_s complex-valued row vector. N_s is the number of scatterers specified by `numscat`. The gains apply to the corresponding paths. Units are dimensionless.

Example: `exp(1i*pi/3)`

Data Types: `double`

Output Arguments

`chmat` — MIMO channel matrix
N_t-by-N_r complex-valued matrix

MIMO channel matrix, returned as an N_t-by-N_r complex-valued matrix. N_t is the number of elements in the transmitting array. N_r is the number of elements in the receiving array.

Data Types: `double`

Version History

Introduced in R2017a

References

[1] Heath, R. Jr. et al. “An Overview of Signal Processing Techniques for Millimeter Wave MIMO Systems”, arXiv.org:1512.03007 [cs.IT], 2015.

[2] Tse, D. and P. Viswanath, *Fundamentals of Wireless Communications*, Cambridge: Cambridge University Press, 2005.

[3] Paulraj, A. *Introduction to Space-Time Wireless Communications*, Cambridge: Cambridge University Press, 2003.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.

See Also

Functions
`blkdiagbfweights` | `diagbfweights` | `waterfill`

Objects
`phased.ScatteringMIMOChannel`

speed2dop

Convert speed to Doppler shift

Syntax

```
dps = speed2dop(radvel,lambda)
```

Description

`dps = speed2dop(radvel,lambda)` returns the one-way Doppler shift in hertz corresponding to the radial velocity `radvel` for the wavelength `lambda`.

Examples

Calculate Doppler Shift from Speed

Calculate the Doppler shift in hertz for a given carrier wavelength and source speed. The radar frequency is 24.15 GHz. Assume a radial speed of 35.76 m/s.

```
radvel = 35.76;
f0 = 24.15e9;
lambda = physconst('LightSpeed')/f0;
doppler_shift = speed2dop(radvel,lambda)
```

```
doppler_shift = 2.8807e+03
```

Input Arguments

`radvel` — Radial velocity
scalar | vector | matrix

Radial velocity in meters per second, specified as a scalar, vector, or matrix.

Data Types: `double`

`lambda` — Wavelength
positive scalar

Wavelength in meters, specified as a positive scalar.

Data Types: `double`

Output Arguments

dps — One-way Doppler shift
scalar | vector | matrix

One-way Doppler shift in hertz, returned as a scalar, vector, or matrix.

More About

Doppler-Radial Velocity Relation

The Doppler shift of a source relative to a receiver can be computed from the relative radial velocity between the source and receiver:

$$\Delta f = \frac{V_{s,r}}{\lambda}$$

where Δf is the Doppler shift in hertz, $V_{s,r}$ denotes the radial velocity of the source relative to the receiver, and λ is the carrier frequency wavelength in meters.

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

References

[1] Rappaport, T. *Wireless Communications: Principles & Practices*. Upper Saddle River, NJ: Prentice Hall, 1996.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

See Also

`dop2speed` | `dopsteeringvec`

sph2cartvec

Convert vector from spherical basis components to Cartesian components

Syntax

```
vr = sph2cartvec(vs,az,el)
```

Description

`vr = sph2cartvec(vs,az,el)` converts the components of a vector or set of vectors, `vs`, from their spherical basis representation to their representation in a local Cartesian coordinate system. A spherical basis representation is the set of components of a vector projected into the right-handed spherical basis given by $(\hat{\mathbf{e}}_{az}, \hat{\mathbf{e}}_{el}, \hat{\mathbf{e}}_{R})$. The orientation of a spherical basis depends upon its location on the sphere as determined by azimuth, `az`, and elevation, `el`.

Examples

Cartesian Representation of Azimuthal Vector

Start with a vector in a spherical basis located at 45° azimuth, 45° elevation. The vector points along the azimuth direction. Compute the vector components with respect to Cartesian coordinates.

```
vs = [1;0;0];
vr = sph2cartvec(vs,45,45)
```

```
vr = 3×1

   -0.7071
    0.7071
         0
```

Input Arguments

vs — Vector in spherical basis representation
3-by-1 column vector | 3-by-*N* matrix

Vector in spherical basis representation specified as a 3-by-1 column vector or 3-by-*N* matrix. Each column of `vs` contains the three components of a vector in the right-handed spherical basis $(\hat{\mathbf{e}}_{az}, \hat{\mathbf{e}}_{el}, \hat{\mathbf{e}}_{R})$.

Example: `[4.0; -3.5; 6.3]`

Data Types: `double`
Complex Number Support: Yes

az — Azimuth angle
scalar in range [–180,180]

Azimuth angle specified as a scalar in the closed range [–180,180]. Angle units are in degrees. To define the azimuth angle of a point on a sphere, construct a vector from the origin to the point. The azimuth angle is the angle in the *xy*-plane from the positive *x*-axis to the vector's orthogonal projection into the *xy*-plane. As examples, zero azimuth angle and zero elevation angle specify a point on the *x*-axis while an azimuth angle of 90° and an elevation angle of zero specify a point on the *y*-axis.

Example: `45`

Data Types: `double`

`el` — Elevation angle
scalar in range [–90,90]

Elevation angle specified as a scalar in the closed range [–90,90]. Angle units are in degrees. To define the elevation of a point on the sphere, construct a vector from the origin to the point. The elevation angle is the angle from its orthogonal projection into the *xy*-plane to the vector itself. As examples, zero elevation angle defines the equator of the sphere and ±90° elevation define the north and south poles, respectively.

Example: `30`

Data Types: `double`

Output Arguments

`vr` — Vector in Cartesian representation
3-by-1 column vector | 3-by-*N* matrix

Cartesian vector returned as a 3-by-1 column vector or 3-by-*N* matrix having the same dimensions as `vs`. Each column of `vr` contains the three components of the vector in the right-handed *x*,*y*,*z* basis.

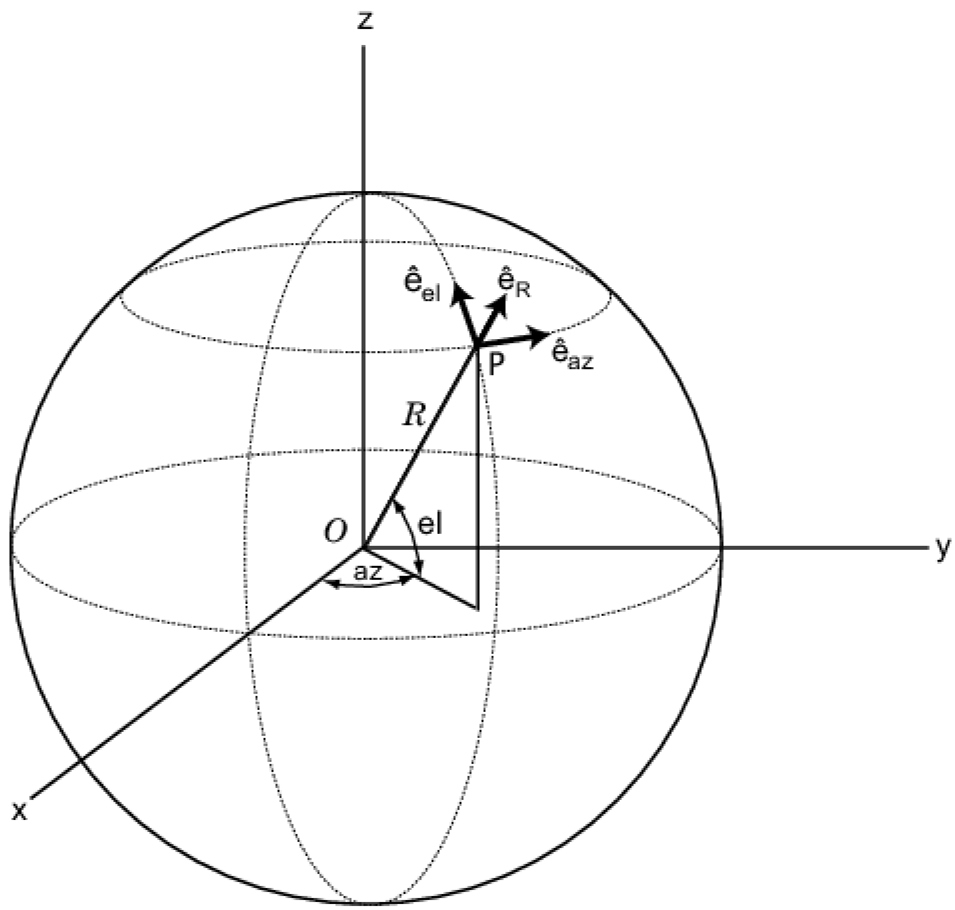
More About

Spherical basis representation of vectors

Spherical basis vectors are a local set of basis vectors which point along the radial and angular directions at any point in space.

The spherical basis is a set of three mutually orthogonal unit vectors $(\hat{\mathbf{e}}_{az}, \hat{\mathbf{e}}_{el}, \hat{\mathbf{e}}_{R})$ defined at a point on the sphere. The first unit vector points along lines of azimuth at constant radius and elevation. The second points along the lines of elevation at constant azimuth and radius. Both are tangent to the surface of the sphere. The third unit vector points radially outward.

The orientation of the basis changes from point to point on the sphere but is independent of *R* so as you move out along the radius, the basis orientation stays the same. The following figure illustrates the orientation of the spherical basis vectors as a function of azimuth and elevation:



For any point on the sphere specified by *az* and *el*, the basis vectors are given by:

$$\begin{aligned}
\widehat{\mathbf{e}}_{\mathbf{az}} &= -\sin(az)\,\widehat{\mathbf{i}} + \cos(az)\,\widehat{\mathbf{j}} \\
\widehat{\mathbf{e}}_{\mathbf{el}} &= -\sin(el)\cos(az)\,\widehat{\mathbf{i}} - \sin(el)\sin(az)\,\widehat{\mathbf{j}} + \cos(el)\,\widehat{\mathbf{k}} \\
\widehat{\mathbf{e}}_{\mathbf{R}} &= \cos(el)\cos(az)\,\widehat{\mathbf{i}} + \cos(el)\sin(az)\,\widehat{\mathbf{j}} + \sin(el)\,\widehat{\mathbf{k}} \quad .
\end{aligned}$$

Any vector can be written in terms of components in this basis as $\mathbf{v} = v_{az}\widehat{\mathbf{e}}_{\mathbf{az}} + v_{el}\widehat{\mathbf{e}}_{\mathbf{el}} + v_R\widehat{\mathbf{e}}_{\mathbf{R}}$. The transformations between spherical basis components and Cartesian components take the form

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} -\sin(az) & -\sin(el)\cos(az) & \cos(el)\cos(az) \\ \cos(az) & -\sin(el)\sin(az) & \cos(el)\sin(az) \\ 0 & \cos(el) & \sin(el) \end{bmatrix} \begin{bmatrix} v_{az} \\ v_{el} \\ v_R \end{bmatrix}$$

.

and

$$\begin{bmatrix} v_{az} \\ v_{el} \\ v_R \end{bmatrix} = \begin{bmatrix} -\sin(az) & \cos(az) & 0 \\ -\sin(el)\cos(az) & -\sin(el)\sin(az) & \cos(el) \\ \cos(el)\cos(az) & \cos(el)\sin(az) & \sin(el) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}.$$

Version History

Introduced in R2013a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`azelaxes` | `cart2sphvec`

spsmooth

Spatial smoothing

Syntax

```
RSM = spsmooth(R,L)
RSM = spsmooth(R,L,'fb')
```

Description

`RSM = spsmooth(R,L)` computes an averaged spatial covariance matrix, `RSM`, from the full spatial covariance matrix, `R`, using spatial smoothing (see Van Trees [1], p. 605). Spatial smoothing creates a smaller averaged covariance matrix over *L* maximum overlapped subarrays. *L* is a positive integer less than *N*. The resulting covariance matrix, `RSM`, has dimensions (*N*–*L*+1)-by-(*N*–*L*+1). Spatial smoothing is useful when two or more signals are correlated.

`RSM = spsmooth(R,L,'fb')` computes an averaged covariance matrix and at the same time performing forward-backward averaging. This syntax can use any of the input arguments in the previous syntax.

Examples

Comparison of Smoothed and Nonsmoothed Covariance Matrices

Construct a 10-element half-wavelength-spaced uniform line array receiving two plane waves arriving from 0° and -25° azimuth. Both elevation angles are 0°. Assume the two signals are partially correlated. The SNR for each signal is 5 dB. The noise is spatially and temporally Gaussian white noise. First, create the spatial covariance matrix from the signal and noise. Then, solve for the number of signals, using `rootmusicdoa`. Next, perform spatial smoothing on the covariance matrix, using `spsmooth`, and solve for the signal arrival angles again using `rootmusicdoa`.

Set up the array and signals. Then, generate the spatial covariance matrix for the array from the signals and noise.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25];
ac = [1 1/5];
scov = ac'*ac;
R = sensorcov(elementPos,angles,db2pow(-5),scov);
```

Solve for the arrival angles using the original covariance matrix.

```
Nsig = 2;
doa =  rootmusicdoa(R,Nsig)
```

```
doa = 1×2
```

```
    0.3603   79.2382
```

The solved-for arrival angles are wrong - they do not agree with the known angles of arrival used to create the covariance matrix.

Next, solve for the arrival angles using a smoothed covariance matrix. Perform spatial smoothing to detect L-1 coherent signals. Choose L = 3.

```
Nsig = 2;
L = 2;
RSM = spsmooth(R,L);
doasm = rootmusicdoa(RSM,Nsig)
```

```
doasm = 1×2

  -25.0000    0.0000
```

In this case, computed angles do agree with the known angles of arrival.

Input Arguments

R — Spatial covariance matrix

complex-valued positive-definite *N*-by-*N* matrix.

Spatial covariance matrix, specified as a complex-valued, positive-definite *N*-by-*N* matrix. In this matrix, *N* represents the number of sensor elements.

Example: [4.3162, –0.2777 –0.2337i; –0.2777 + 0.2337i , 4.3162]

Data Types: `double`
Complex Number Support: Yes

L — Maximum number of overlapped subarrays

positive integer

Maximum number of overlapped subarrays, specified as a positive integer. The value *L* must be less than the number of sensors, *N*.

Example: 2

Data Types: `double`

Output Arguments

RSM — Smoothed covariance matrix

complex-valued *M*-by-*M* matrix

Smoothed covariance matrix, returned as a complex-valued, *M*-by-*M* matrix. The dimension *M* is given by $M = N–L+1$.

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York, NY: Wiley-Interscience, 2002.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`aictest` | `espritdoa` | `mdltest` | `rootmusicdoa`

steervec

Steering vector

Syntax

```
sv = steervec(pos,ang)
sv = steervec(pos,ang,nqbits)
```

Description

`sv = steervec(pos,ang)` returns the steering vector `sv` for each incoming plane wave or set of plane waves impinging on a sensor array. The steering vector represents the set of phase-delays for an incoming wave at each sensor element. The `pos` argument specifies the positions of the sensor array elements. The `ang` argument specifies the incoming wave arrival directions in terms of azimuth and elevation angles. The steering vector, `sv`, is an N-by-M complex-valued matrix. In this matrix, N represents the number of element positions in the sensor array while M represents the number of incoming waves. Each column of `sv` contains the steering vector for the corresponding direction specified in `ang`. All elements in the sensor array are assumed to be isotropic.

`sv = steervec(pos,ang,nqbits)` returns quantized narrowband steering vector when the number of phase shifter bits is set to `nqbits`.

Examples

Line Array Steering Vector

Specify a uniform line array of five elements spaced 10 cm apart. Then, specify an incoming plane wave with a frequency of 1 GHz and an arrival direction of 45° azimuth and 0° elevation. Compute the steering vector of this wave.

```
elementPos = (0:.1:.4);
c = physconst('LightSpeed');
fc = 1e9;
lam = c/fc;
ang = [45;0];
sv = steervec(elementPos/lam,ang)
```

```
sv = 5×1 complex

   1.0000 + 0.0000i
   0.0887 + 0.9961i
  -0.9843 + 0.1767i
  -0.2633 - 0.9647i
   0.9376 - 0.3478i
```

Quantized Line Array Steering Vector

Specify a uniform line array (ULA) containing five isotropic elements spaced 10 cm apart. Then, specify an incoming plane wave having a frequency of 1 GHz and an arrival direction of 45° azimuth and 0° elevation. Compute the steering vector of this wave. Quantize the steering vector to three bits.

```
elementPos = (0:.1:.4);
c = physconst('LightSpeed');
fc = 1e9;
lam = c/fc;
ang = [45;0];
sv = steervec(elementPos/lam,ang,3)
```

```
sv = 5×1 complex

   1.0000 + 0.0000i
   0.0000 + 1.0000i
  -1.0000 + 0.0000i
  -0.0000 - 1.0000i
   1.0000 + 0.0000i
```

Input Arguments

pos — Positions of array sensor elements
1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array, specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero. When `pos` is a 2-by-*N* matrix, it represents the *(y,z)*-coordinates of the sensor elements of a planar array. This array is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. When `pos` is a 3-by-*N* matrix, then the array can have an arbitrary shape. Sensor positions are in terms of signal wavelength.

Example: `[0,0,0; 0.1,0.4,0.3; 1,1,1]`

Data Types: `double`

ang — Arrival directions of incoming signals
1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Arrival directions of incoming signals specified as a 1-by-*M* vector or a 2-by-*M* matrix, where *M* is the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the incoming signal `[az;el]`. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the arrival direction vector onto the *xy* plane. It is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the arrival direction vector and *xy*-plane. It is positive when measured towards the *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero. Angle units are specified in degrees.

Example: [45;0]

Data Types: `double`

`nqbits` — Number of phase shifter quantization bits
0 (default) | non-negative integer

Number of bits used to quantize the phase shift in beamformer or steering vector weights, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Example: 5

Output Arguments

`sv` — Steering vector
N-by-*M* complex-valued matrix

Steering vector returned as an *N*-by-*M* complex-valued matrix. In this matrix, *N* represents the number of sensor elements of the array and *M* represents the number of incoming plane waves. Each column of `sv` corresponds to the same column in `ang`.

Version History

Introduced in R2013a

References

[1] Van Trees, H.L. *Optimum Array Processing*. New York, NY: Wiley-Interscience, 2002.

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`cbfweights` | `lcmvweights` | `mvdrweights` | `sensorcov` | `phased.SteeringVector`

stokes

Stokes parameters of polarized field

Syntax

```
G = stokes(fv)
stokes(fv)
```

Description

`G = stokes(fv)` returns the four Stokes parameters `G` of a polarized field or set of fields specified in `fv`. The field should be expressed in terms of linear polarization components. The expression of a field in terms of a two-row vector of linear polarization components is called the Jones vector formalism.

`stokes(fv)` displays the Stokes parameters corresponding to `fv` as points on the Poincare sphere.

Examples

Stokes Vector

Create a left circularly-polarized field. Convert it to a linear representation and compute the Stokes vector.

```
cfv = [2;0];
fv = circpol2pol(cfv);
G = stokes(fv)
```

```
G = 4×1

    4.0000
         0
         0
    4.0000
```

Poincare Sphere

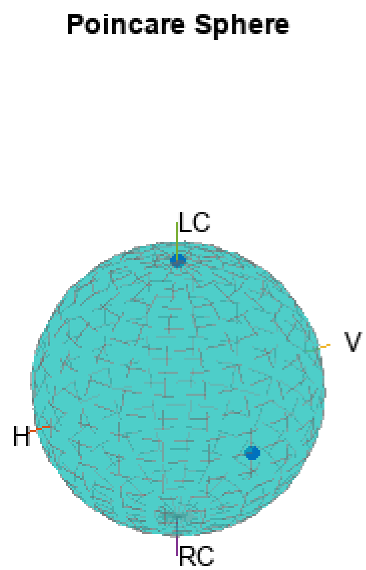
Display points on the Poincare sphere for a left circularly-polarized field and a 45° linear polarized field.

```
fv = [sqrt(2)/2, 1; sqrt(2)/2*1i, 1];
G = stokes(fv)
```

```
G = 4×2

    1.0000    2.0000
         0         0
```

```
         0    2.0000
    1.0000         0
```

```
stokes(fv);
```



The point at the north pole represents the left circularly-polarized field. The point on the equator represents the 45° linear polarized field.

Input Arguments

fv — Field vector in linear polarization representation or linear polarization ratio
1-by-*N* complex-value row vector or 2-by-*N* complex-value matrix

Field vector in its linear polarization representation specified as a 2-by-*N* complex-valued matrix or in its linear polarization ratio representation specified as a 1-by-*N* complex-valued row vector. If `fv` is a matrix, each column of `fv` represents a field in the form `[Eh;Ev]`, where `Eh` and `Ev` are its horizontal and vertical linear polarization components. The expression of a field in terms of a two-row vector of linear polarization components is called the Jones vector formalism. If `fv` is a vector, each entry in `fv` is contains the polarization ratio, `Ev/Eh`.

Example: [sqrt(2)/2*1i; 1]

Data Types: `double`
Complex Number Support: Yes

Output Arguments

G — Stokes parameters
4-by-*N* matrix of Stokes parameters.

G contains the four Stokes parameters for each polarized field specified in fv. The Stokes parameters are computed from combinations of intensities of the field:

- G_0 describes the total intensity of the field.
- G_1 describes the preponderance of horizontal linear polarization intensity over vertical linear polarization intensity.
- G_2 describes the preponderance of +45° linear polarization intensity over -45° linear polarization intensity.
- G_3 describes the preponderance of right circular polarization intensity over left circular polarization intensity.

Version History

Introduced in R2013a

References

[1] Mott, H., *Antennas for Radar and Communications*, John Wiley & Sons, 1992.

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302.

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.
- Supported only when output arguments are specified.

See Also

circpol2pol | pol2circpol | polellip | polratio

stretchfreq2rng

Convert frequency offset to range

Syntax

```
R = stretchfreq2rng(FREQ,SLOPE,REFRNG)
R = stretchfreq2rng(FREQ,SLOPE,REFRNG,V)
```

Description

`R = stretchfreq2rng(FREQ,SLOPE,REFRNG)` returns the range corresponding to the frequency offset `FREQ`. The computation assumes you obtained `FREQ` through stretch processing with a reference range of `REFRNG`. The sweeping slope of the linear FM waveform is `SLOPE`.

`R = stretchfreq2rng(FREQ,SLOPE,REFRNG,V)` specifies the propagation speed `V`.

Examples

Range Corresponding to Frequency Offset

Calculate the range corresponding to a frequency offset of 2 kHz obtained from stretch processing. Assume the reference range is 5000 m and the linear FM waveform has a sweeping slope of 2 GHz/s.

```
r = stretchfreq2rng(2e3,2e9,5000)
```

```
r = 4.8501e+03
```

Input Arguments

`FREQ` — Frequency offset
scalar | vector

Frequency offset in hertz, specified as a scalar or vector.

Data Types: `double`

`SLOPE` — Sweeping slope of the linear FM waveform
nonzero scalar

Sweeping slope of the linear FM waveform, in hertz per second, specified as a nonzero scalar.

Data Types: `double`

`REFRNG` — Reference range
scalar

Reference range in meters, specified as a scalar.

Data Types: `double`

V — Propagation speed
`physconst("LightSpeed")` (default) | positive scalar

Propagation speed in meters per second, specified as a positive scalar.

Data Types: `double`

Output Arguments

R — Range
scalar | vector

Range in meters, returned as a scalar or vector. `R` has the same dimensions as `FREQ` .

Version History

Introduced in R2012a

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`phased.LinearFMWaveform` | `phased.StretchProcessor` | `ambgfun` | `beat2range` | `range2beat` | `rdcoupling`

Topics
"Range Estimation Using Stretch Processing"
"Stretch Processing"

systemp

Receiver system-noise temperature

Syntax

```
STEMP = systemp(NF)
STEMP = systemp(NF,REFTEMP)
```

Description

`STEMP = systemp(NF)` calculates the effective system-noise temperature, `STEMP`, in kelvin, based on the noise figure, `NF`, in decibels. The reference temperature is 290 K.

`STEMP = systemp(NF,REFTEMP)` specifies the reference temperature.

Examples

Compute System Noise Temperature

Calculate the system noise temperature of a receiver with a 300 K reference temperature and a 5 dB noise figure.

```
T = systemp(5,300)
```

```
T = 948.6833
```

Input Arguments

NF — Noise figure
nonnegative scalar

Noise figure in decibels, specified as a nonnegative scalar. The noise figure is the ratio of the actual output noise power in a receiver to the noise power output of an ideal receiver.

Data Types: `double`

REFTEMP — Reference temperature
`290` (default) | nonnegative scalar

Reference temperature in kelvins, specified as a nonnegative scalar. The output of an ideal receiver has a white noise power spectral density that is approximately the Boltzmann constant times the reference temperature in kelvin.

Data Types: `double`

Output Arguments

STEMP — Effective system-noise temperature
nonnegative scalar

Effective system-noise temperature in kelvins, returned as a nonnegative scalar. The effective system-noise temperature is `REFTEMP*10^(NF/10)`.

Version History

Introduced in R2011a

References

[1] Skolnik, M. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`noisepow` | `phased.ReceiverPreamp`

taylortaperc

Taylor nbar taper for arrays

Syntax

```
W = taylortaperc(pos,diam)
W = taylortaperc(pos,diam,nbar)
W = taylortaperc(pos,diam,nbar,sll)
W = taylortaperc(pos,diam,nbar,sll,cpos)
```

Description

`W = taylortaperc(pos,diam)` returns the value of a Taylor n-bar taper, `W`, at sensor element positions specified by `pos` in a circular aperture having diameter `diam`.

`W = taylortaperc(pos,diam,nbar)` also specifies, `nbar`, the number of approximately constant-level sidelobes next to the mainlobe.

`W = taylortaperc(pos,diam,nbar,sll)` also specifies the maximum sidelobe level, `sll`, relative to the mainlobe peak.

`W = taylortaperc(pos,diam,nbar,sll,cpos)` also specifies the center of the array, `cpos`. Without this argument, the function sets the array center to the computed centroid of the array.

Examples

Default Taylor Taper Circular Array

Apply a Taylor nbar taper to a circular aperture array. Obtain the circular aperture by cropping a square uniform rectangular array into a circle. Let all the parameters remain at their default values: nbar is 4 and the sidelobe level is –30. Let the center of the array be the centroid of the array elements. Plot the array power pattern at 300 MHz.

Create a square URA with a side length of 10 m. Set the element spacing to 0.5 m. The spacing is equal to one-half wavelength at this frequency.

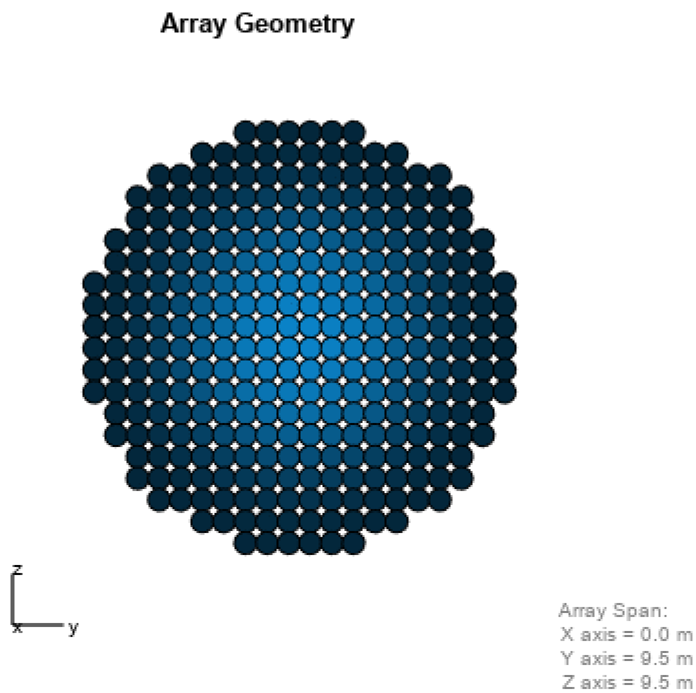
```
fc = 300.0e6;
diam = 10.0;
d = 0.5;
nelem = ceil(diam/d);
pos = getElementPosition(phased.URA(nelem,d));
```

Use the `phased.ConformalArray` System object™ to model a circular array. Create a circular array by removing all elements outside a radius one-half the side-length of the URA. Then apply the Taylor nbar tapering to the array.

```
pos(:,sum(pos.^2) > (diam/2)^2) = [];
antenna = phased.ConformalArray('ElementPosition',pos);
antenna.Taper = taylortaperc(pos,diam);
```

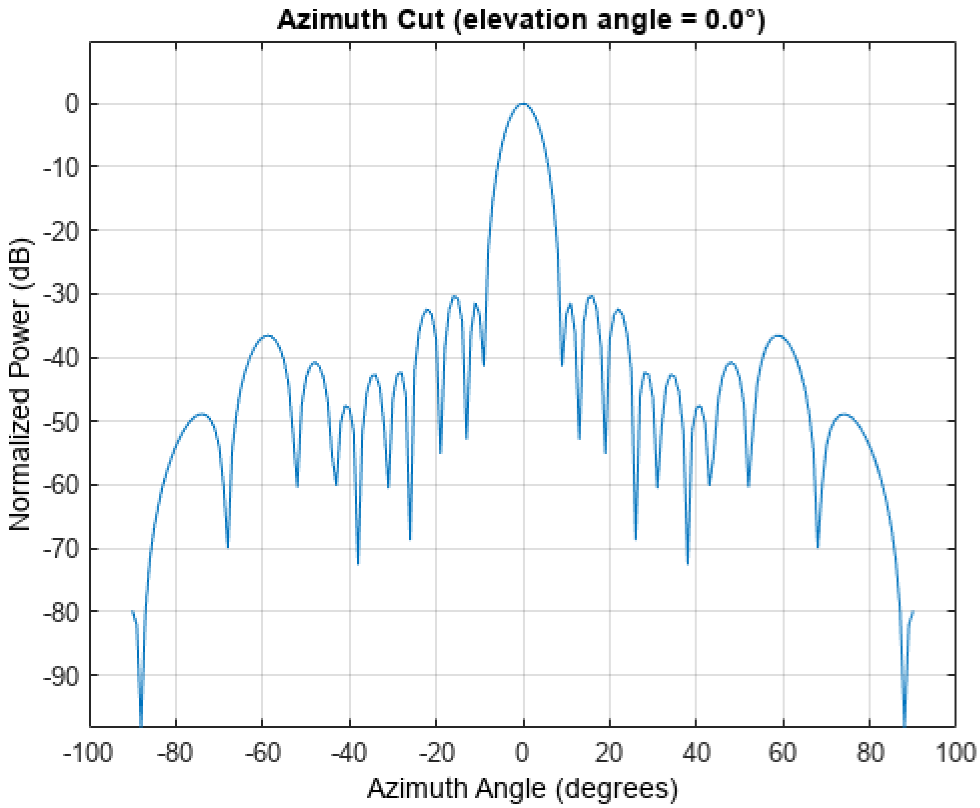
View the array.

```
viewArray(antenna,'ShowTaper',true)
```



Display the array power pattern as a function of azimuth angle.

```
clf
pattern(antenna,fc,-90:1:90,0,'CoordinateSystem','rectangular','Type','powerdb')
```



Taylor Taper Circular Array Specifying Nbar

Apply a Taylor nbar taper to a circular aperture array. Create the circular aperture by cropping a square uniform rectangular array into a circle. Set the value of nbar to 2. Let the sidelobe level assume a default value of –30. Let the center of the array be the centroid of the array elements. Plot the array power pattern at 300 MHz.

Create a square URA with a side length of 10 m. Set the element spacing to 0.5 m. The spacing is equal to one-half wavelength at this frequency.

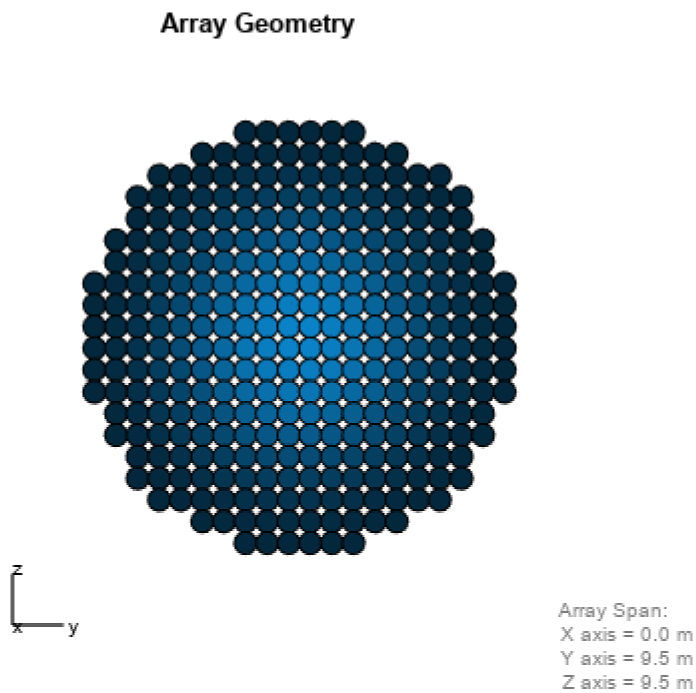
```
fc = 300.0e6;
diam = 10.0;
d = 0.5;
nbar = 2;
nelem = ceil(diam/d);
pos = getElementPosition(phased.URA(nelem,d));
```

Use the `phased.ConformalArray` System object™ to model a circular array. Create a circular array by removing all elements outside a radius one-half the side-length of the URA. Then apply the Taylor nbar tapering to the array.

```
pos(:,sum(pos.^2) > (diam/2)^2) = [];
antenna = phased.ConformalArray('ElementPosition',pos);
antenna.Taper = taylortaperc(pos,diam,nbar);
```

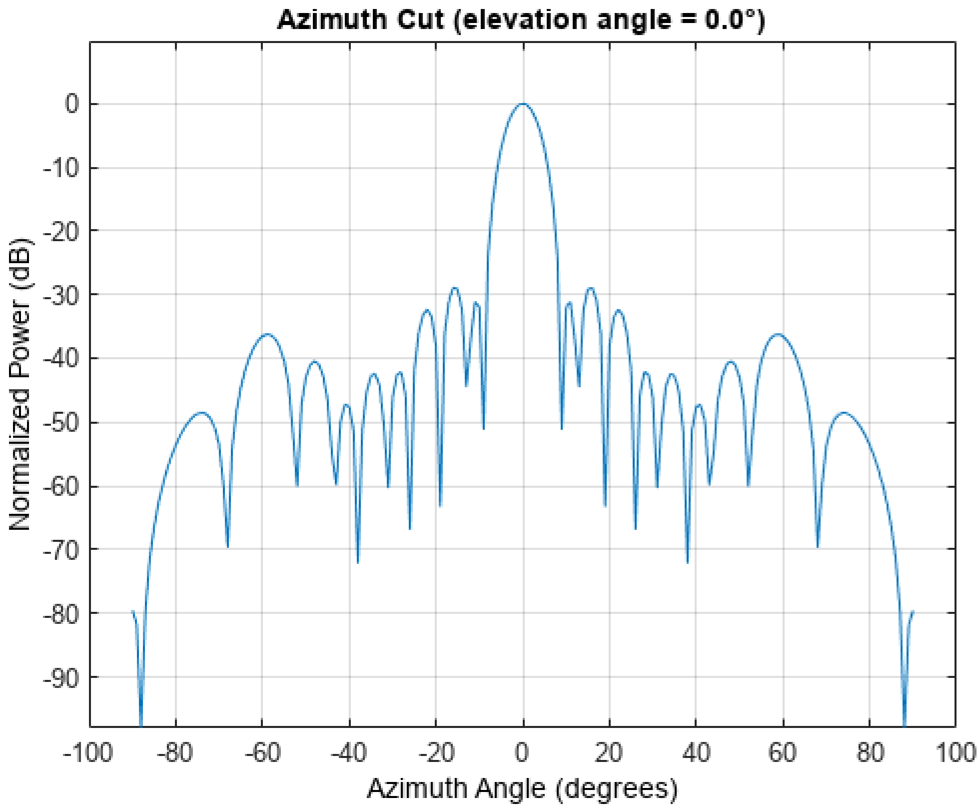
View the array.

```
viewArray(antenna,'ShowTaper',true)
```



Display the array power pattern as a function of azimuth angle.

```
clf
pattern(antenna,fc,-90:1:90,0,'CoordinateSystem','rectangular','Type','powerdb')
```



Taylor Taper Circular Array Specifying Sidelobe Level

Apply a Taylor nbar taper to a circular aperture array. Create the circular aperture by cropping a square uniform rectangular array into a circle. Set the value of nbar to 4. Set the sidelobe level to –25. Let the center of the array be the centroid of the array elements. Plot the array power pattern at 300 MHz.

First, create a square URA with a side length of 10 m. Set the element spacing to 0.5 m. The spacing is equal to one-half wavelength at this frequency.

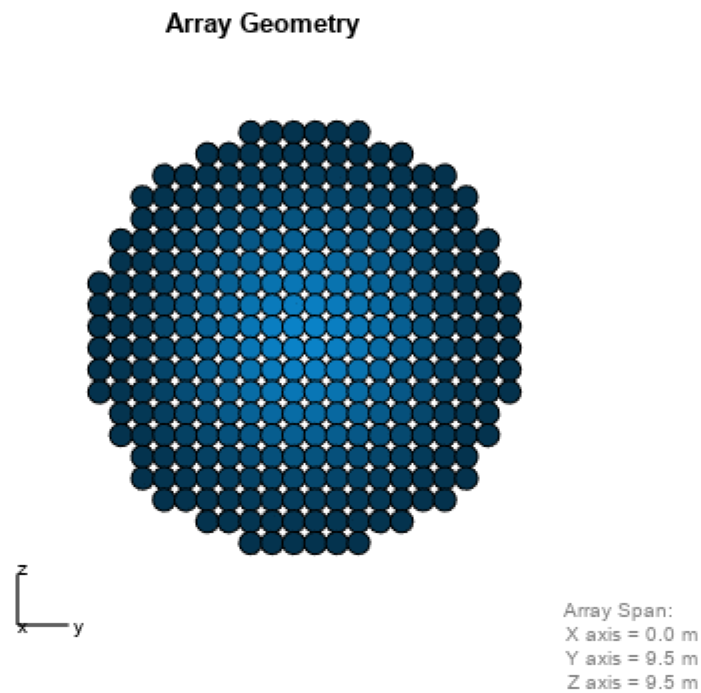
```
fc = 300.0e6;
diam = 10.0;
d = 0.5;
nbar = 2;
sll = -25;
nelem = ceil(diam/d);
pos = getElementPosition(phased.URA(nelem,d));
```

Use the `phased.ConformalArray` System object™ to model a circular array. Create a circular array by removing all elements outside a radius one-half the side-length of the URA. Then apply the Taylor nbar tapering to the array.

```
pos(:,sum(pos.^2) > (diam/2)^2) = [];
antenna = phased.ConformalArray('ElementPosition',pos);
antenna.Taper = taylortaperc(pos,diam,nbar,sll);
```

View the array.

```
viewArray(antenna,'ShowTaper',true)
```



Display the array power pattern as a function of azimuth angle.

```
clf
pattern(antenna,fc,-90:1:90,0,'CoordinateSystem','rectangular','Type','powerdb')
```

Taylor Taper Circular Array Specifying Array Center

Apply a Taylor nbar taper to a circular aperture array. Create the circular aperture by cropping a square uniform rectangular array into a circle. Set the sidelobe level to –25. Set the center of the array to the origin. Plot the array power pattern at 300 MHz.

Create a square URA with a side length of 10 m. Set the element spacing to 0.5 m. The spacing is equal to one-half wavelength at this frequency.

```
fc = 300.0e6;
diam = 10.0;
d = 0.5;
sll = -25;
```

Compute nbar from the sidelobe level.

```
A = acosh(10^(-sll/20))/pi;
nbar = ceil(2*A^2 + 0.5)
```

```
nbar = 4
```

Create the URA element positions.

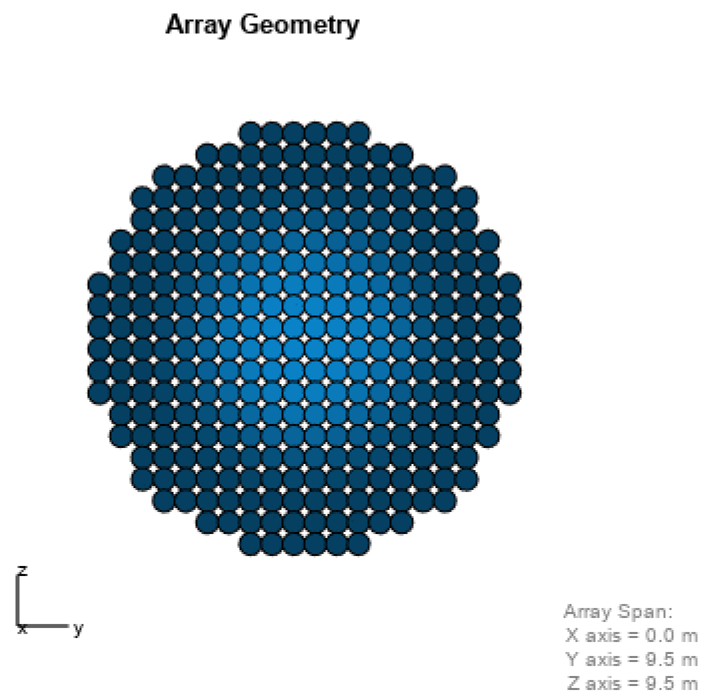
```
cpos = [0;0;0];
nelem = ceil(diam/d);
pos = getElementPosition(phased.URA(nelem,d));
```

Use the `phased.ConformalArray` System object™ to model a circular array. Create a circular array by removing all elements outside a radius one-half the side-length of the URA. Then apply the Taylor nbar tapering to the array.

```
pos(:,sum(pos.^2) > (diam/2)^2) = [];
antenna = phased.ConformalArray('ElementPosition',pos);
antenna.Taper = taylortaperc(pos,diam,nbar,sll,cpos);
```

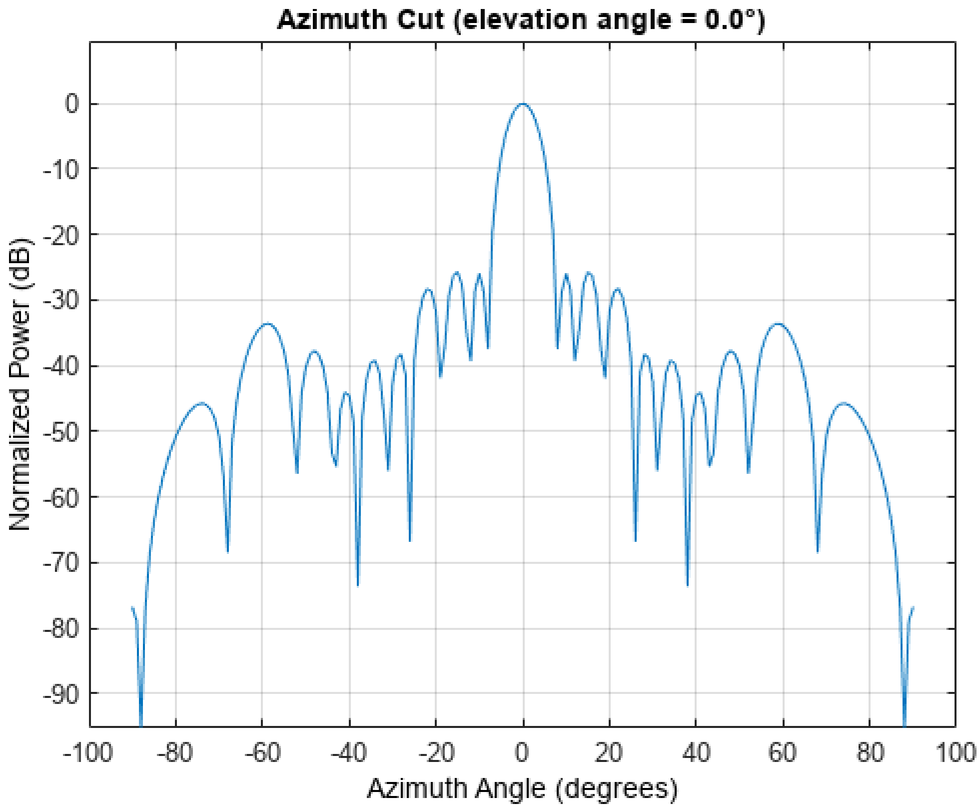
View the array.

```
viewArray(antenna,'ShowTaper',true)
```



Display the array power pattern as a function of azimuth angle.

```
clf
pattern(antenna,fc,-90:1:90,0,'CoordinateSystem','rectangular','Type','powerdb')
```



Input Arguments

`pos` — Position of array elements

2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Position of array elements, specified as a 2-by-*N* or 3-by-*N* real-valued matrix where *N* is the number of elements. If `pos` is a 2-by-*N* matrix, then all elements lie in the *z = 0* plane. Each column specifies the position, `[x;y]`, of the element. If `pos` is a 3-by-*N* matrix, its columns represent the positions of array elements in `[x;y;z]` format. `W` is an *N*-by-1 column vector containing the Taylor tapers. The 2-by-*N* form is designed for planar arrays although you can use the 3-by-*N* form and set the third row to zero. Position units are in meters.

Example: `[–5,–5,5,5;-5,5,5,-5]`

Data Types: `double`

`diam` — Array diameter

positive scalar

Array diameter, specified as a positive scalar. Diameter units are in meters.

Example: `15.5`

Data Types: `double`

`nbar` — Number of nearly equal sidelobes

4 (default) | positive integer

Number of nearly equal sidelobes on each side of the mainlobe, specified as a positive integer. Units are dimensionless.

Example: 3

Data Types: `double`

`sll` — Maximum sidelobe level
`-30.0` (default) | negative scalar

Maximum sidelobe, specified as a negative scalar. Sidelobe levels are referenced to the mainlobe. Units are in dB.

Example: `-10.0`

Data Types: `double`

`cpos` — Array center
array centroid (default) | real-valued 2-by-1 vector | real-valued 3-by-1 vector

Array center, specified as a real-valued 2-by-1 or 3-by-1 vector. Units are in meters. Use a 2-by-1 vector when the element positions are specified as a 2-by-*N* matrix. The default value is the computed centroid of all the array elements.

Example: `[5;-10;3]`

Data Types: `double`

Output Arguments

`W` — Taylor weights
real-valued *N*-by-1 column vector

Taylor weights, returned as a real-valued *N*-by-1 column vector. *N* is the number of array elements. Units are dimensionless.

Algorithms

Compute Minimum Value of N-bar

A useful guideline for choosing a value of `nbar` that meets the required sidelobe level (sll), as specified in the `sll` argument, is to satisfy the inequality

$$\bar{n} \geq \frac{2}{\pi^2}\left(\cosh^{-1}\left(10^{-\frac{sll}{20}}\right)\right)^2 + 0.5$$

This is a recommendation and you may be able to use a smaller value.

Version History

Introduced in R2016b

References

[1] Taylor, T. "Design of Circular Aperture for Narrow Beamwidth and Low Sidelobes." *IRE Trans. on Antennas and Propagation*. Vol. 5, No. 1, January 1960, pp. 17-22.

[2] Van Trees, H. L. *Optimal Array Processing: Part 4 of Detection, Estimation, and Modulation Theory.* New York: A. J. Wiley & Sons, Inc., 2002.

[3] Hansen, R. C. "Tables of Taylor Distributions for Circular Aperture Antennas." *IRE Trans. on Antenna and Propagation.*Vol. 8, No. 1, January 1960, pp. 23-26.

[4] Hansen, R. C. "Array Pattern Control and Synthesis." *Proceedings of the IEEE.* Vol. 80, No. 1, January 1992, pp. 141-151.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`taylorwin`

time2range

Convert propagation time to propagation distance

Syntax

```
r = time2range(t)
r = time2range(t,c)
```

Description

`r = time2range(t)` returns the distance a signal propagates during `t` seconds. The propagation is assumed to be two-way, as in a monostatic radar system.

`r = time2range(t,c)` specifies the signal propagation speed.

Examples

Minimum Detectable Range for Specified Pulse Width

Calculate the minimum detectable range for a monostatic radar system where the pulse width is 2 ms.

```
t = 2e-3;
r = time2range(t)
```

```
r = 2.9979e+05
```

Input Arguments

t — Propagation time
array of positive numbers

Propagation time in seconds, specified as an array of positive numbers.

c — Signal propagation speed
speed of light (default) | positive scalar

Signal propagation speed, specified as a positive scalar in meters per second.

Data Types: `double`

Output Arguments

r — Propagation distance
array of positive numbers

Propagation distance in meters, returned as an array of positive numbers. The dimensions of `r` are the same as those of `t`.

Data Types: `double`

Algorithms

The function computes `c*t/2`.

Version History

Introduced in R2012b

References

[1] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`range2time` | `range2bw` | `phased.FMCWWaveform`

unigrid

Uniform grid

Syntax

```
grd = unigrid(startval,stp,endval)
grd = unigrid(startval,stp,endval,intype)
```

Description

`grd = unigrid(startval,stp,endval)` returns a uniformly sampled grid from the closed interval `[startval,endval]`, starting from `startval`. `stp` specifies the step size. This syntax is the same as calling `startval:stp:endval`.

`grd = unigrid(startval,stp,endval,intype)` specifies whether the interval is closed, or semi-open.

Examples

Create Uniform Grids

Create a uniform closed interval grid with a positive increment.

```
grid = unigrid(0,0.1,1);
grid(1)
```

```
ans = 0
```

```
grid(end)
```

```
ans = 1
```

Note that `grid(1) = 0` and `grid(end) = 1`.

Create a uniform grid with a semi-open interval.

```
grid = unigrid(0,0.1,1,'[)');
grid(1)
```

```
ans = 0
```

```
grid(end)
```

```
ans = 0.9000
```

In this case, `grid(end) = 0.9`

Create a decreasing grid with a semi-open interval.

```
grid = unigrid(1,-0.2,0,'[)')
```

```
grid = 1×5
```

```
    1.0000    0.8000    0.6000    0.4000    0.2000
```

Input Arguments

`startval` — Start value
real scalar

Start value of the uniform grid, specified as a real scalar.

Data Types: `double`

`endval` — End value
real scalar

End value of the uniform grid, specified as a real scalar.

Data Types: `double`

`stp` — Step size
real scalar

Step size, specified as a real scalar

Data Types: `double`

`intype` — Interval type
`"[]"` (default) | `"[)"`

Interval type, specified as `"[]"`, which represents a closed interval, or `"[)"`, which represents a semi-open interval. Specifying a closed interval does not always cause `grd` to contain the value `endval`. The inclusion of `endval` in a closed interval also depends on the step size `stp`.

Data Types: `char` | `string`

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`linspace` | `val2ind`

uv2azel

Convert u/v coordinates to azimuth/elevation angles

Syntax

```
AzEl = uv2azel(UV)
```

Description

`AzEl = uv2azel(UV)` converts the *u*/*v* space on page 2-482 coordinates to their corresponding azimuth/elevation angle on page 2-483 pairs.

Examples

Conversion of U/V Coordinates to AzEl

Find the corresponding azimuth/elevation representation for $u = 0.5$ and $v = 0$.

```
azel = uv2azel([0.5; 0])
```

```
azel = 2×1

   30.0000
         0
```

Input Arguments

UV — Angle in u/v space

two-row matrix

Angle in *u*/*v* space, specified as a two-row matrix. Each column of the matrix represents a pair of coordinates in the form [*u*; *v*]. Each coordinate is between –1 and 1, inclusive. Also, each pair must satisfy $u^2 + v^2 \leq 1$.

Data Types: `double`

Output Arguments

`AzEl` — Azimuth/elevation angle pairs

two-row matrix

Azimuth and elevation angles, returned as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [azimuth; elevation]. The matrix dimensions of `AzEl` are the same as those of `UV`.

More About

U/V Space

The u/v coordinates for the positive hemisphere $x \geq 0$ can be derived from the phi and theta angles on page 2-482.

The relation between the two coordinates is

$$u = \sin\theta\cos\phi$$
$$v = \sin\theta\sin\phi$$

In these expressions, φ and θ are the phi and theta angles, respectively.

To convert azimuth and elevation to u and v use the transformation

$$u = \cos el \sin az$$
$$v = \sin el$$

which is valid only in the range *abs(az)≤=90*.

The values of u and v satisfy the inequalities

$$-1 \leq u \leq 1$$
$$-1 \leq v \leq 1$$
$$u^2 + v^2 \leq 1$$

Conversely, the phi and theta angles can be written in terms of u and v using

$$\tan\phi = v/u$$
$$\sin\theta = \sqrt{u^2 + v^2}$$

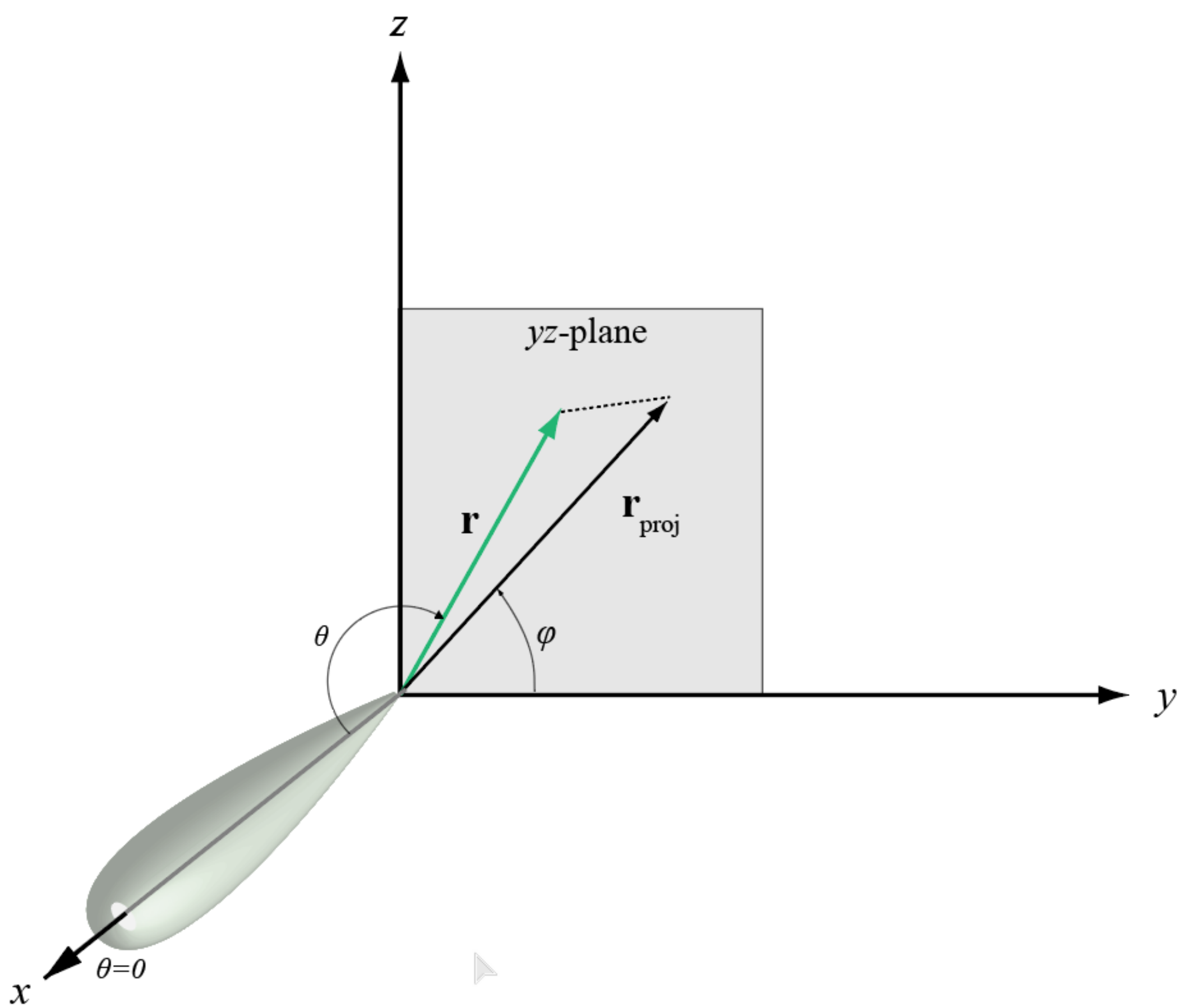
The azimuth and elevation angles can also be written in terms of u and v:

$$\sin el = v$$
$$\tan az = \frac{u}{\sqrt{1 - u^2 - v^2}}$$

Phi Angle, Theta Angle

The phi angle (φ) is the angle from the positive y-axis to the vector's orthogonal projection onto the yz plane. The angle is positive toward the positive z-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the x-axis to the vector itself. The angle is positive toward the yz plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi\sin\theta$

$\tan az = \cos\phi\tan\theta$

$\cos\theta = \cos el\cos az$

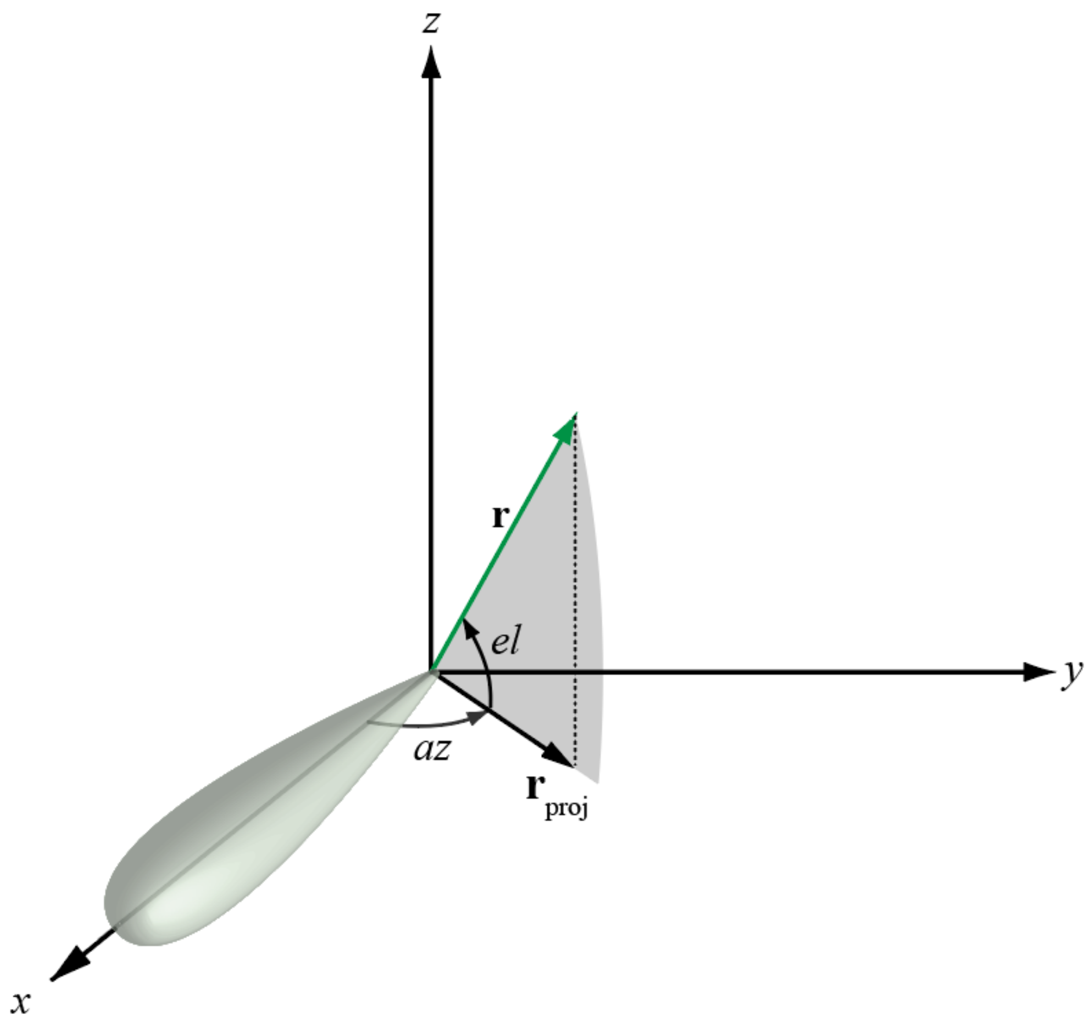
$\tan\phi = \tan el/\sin az$

Azimuth Angle, Elevation Angle

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

`azel2uv`

Topics

"Spherical Coordinates"

uv2azelpat

Convert radiation pattern from u/v form to azimuth/elevation form

Syntax

```
pat_azel = uv2azelpat(pat_uv,u,v)
pat_azel = uv2azelpat(pat_uv,u,v,az,el)
[pat_azel,az_pat,el_pat] = uv2azelpat( ___ )
```

Description

`pat_azel = uv2azelpat(pat_uv,u,v)` expresses the antenna radiation pattern `pat_azel` in azimuth/elevation angle on page 2-491 coordinates instead of u/v space on page 2-490 coordinates. `pat_uv` samples the pattern at *u* angles in `u` and *v* angles in `v`. The `pat_azel` matrix uses a default grid that covers azimuth values from –90 to 90 degrees and elevation values from –90 to 90 degrees. In this grid, `pat_azel` is uniformly sampled with a step size of 1 for azimuth and elevation. The function interpolates to estimate the response of the antenna at a given direction.

`pat_azel = uv2azelpat(pat_uv,u,v,az,el)` uses vectors `az` and `el` to specify the grid at which to sample `pat_azel`. To avoid interpolation errors, `az` should cover the range [–90, 90] and `el` should cover the range [–90, 90].

`[pat_azel,az_pat,el_pat] = uv2azelpat( ___ )` returns vectors containing the azimuth and elevation angles at which `pat_azel` samples the pattern, using any of the input arguments in the previous syntaxes.

Examples

Convert Radiation Pattern

Convert a radiation pattern to azimuth/elevation form with the angles spaced 1° apart.

Define the pattern in terms of *u* and *v*. Because *u* and *v* values outside the unit circle are not physical, set the pattern values in this region to zero.

```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Convert the pattern to azimuth/elevation space.

```
pat_azel = uv2azelpat(pat_uv,u,v);
```

Plot Converted Radiation Pattern

Convert a radiation pattern to azimuth/elevation form with the angles spaced 1° apart.

Define the pattern in terms of *u* and *v*. Because *u* and *v* values outside the unit circle are not physical, set the pattern values in this region to zero.

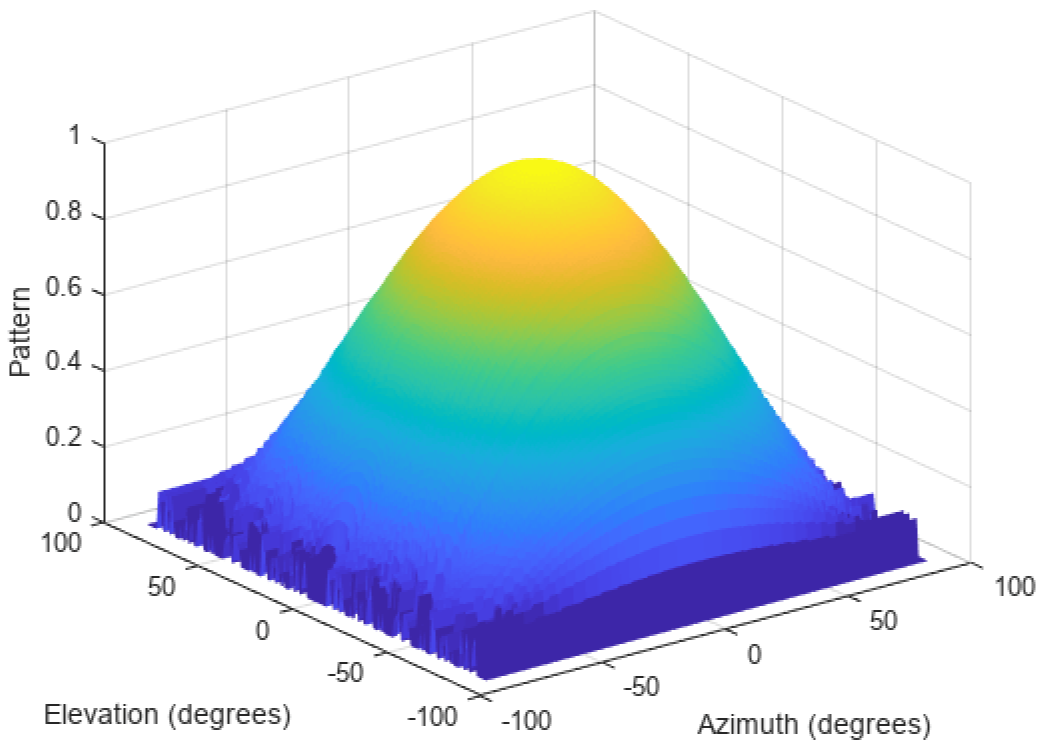
```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Convert the pattern to azimuth/elevation space. Store the azimuth and elevation angles for plotting.

```
[pat_azel,az,el] = uv2azelpat(pat_uv,u,v);
```

Plot the pattern.

```
H = surf(az,el,pat_azel);
H.LineStyle = 'none';
xlabel('Azimuth (degrees)')
ylabel('Elevation (degrees)')
zlabel('Pattern')
```



Convert Radiation Pattern Using Specific Azimuth/Elevation Values

Convert a radiation pattern to azimuth/elevation form, with the angles spaced 5° apart.

Define the pattern in terms of *u* and *v*. Because *u* and *v* values outside the unit circle are not physical, set the pattern values in this region to zero.

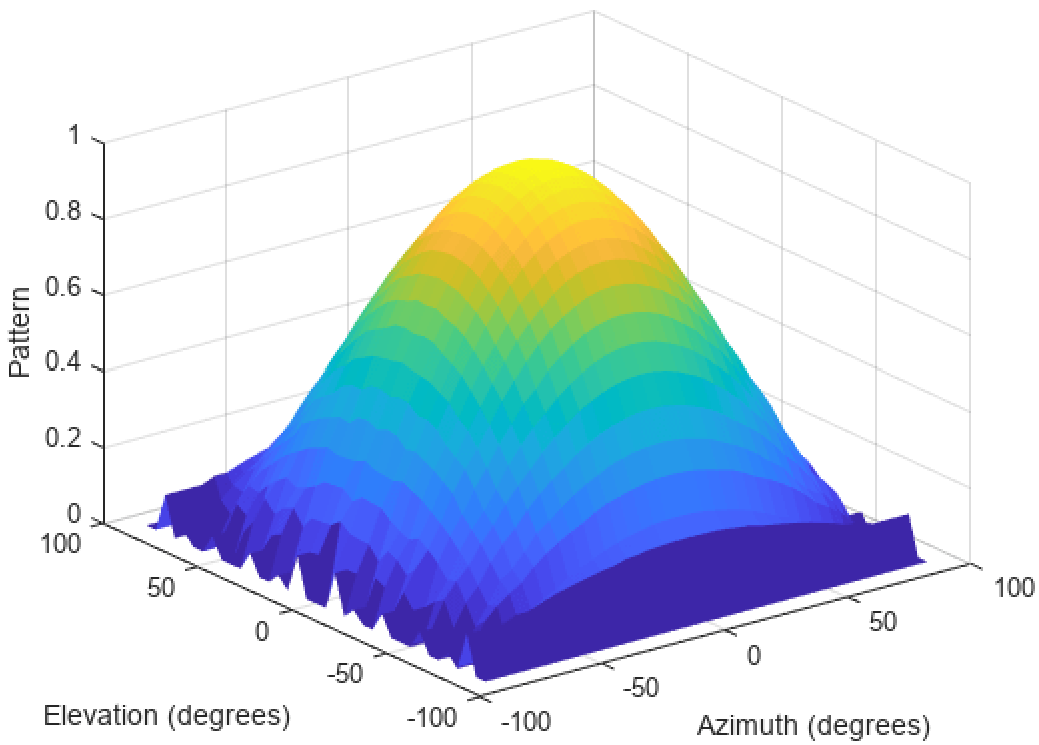
```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Define the set of azimuth and elevation angles at which to sample the pattern. Then convert the pattern.

```
az = -90:5:90;
el = -90:5:90;
pat_azel = uv2azelpat(pat_uv,u,v,az,el);
```

Plot the pattern.

```
H = surf(az,el,pat_azel);
H.LineStyle = 'none';
xlabel('Azimuth (degrees)')
ylabel('Elevation (degrees)')
zlabel('Pattern')
```



Input Arguments

pat_uv — Antenna radiation pattern in *u*/*v* form
Q-by-P matrix

Antenna radiation pattern in *u*/*v* form, specified as a Q-by-P matrix. pat_uv samples the 3-D magnitude pattern in decibels in terms of *u* and *v* coordinates. P is the length of the u vector and Q is the length of the v vector.

Data Types: double

u — *u* coordinates
vector of length P

u coordinates at which pat_uv samples the pattern, specified as a vector of length P. Each coordinate is between –1 and 1.

Data Types: double

v — *v* coordinates
vector of length Q

v coordinates at which pat_uv samples the pattern, specified as a vector of length Q. Each coordinate is between –1 and 1.

Data Types: double

az — Azimuth angles
[-90:90] (default) | vector of length L

Azimuth angles at which pat_azel samples the pattern, specified as a vector of length L. Each azimuth angle is in degrees, between –90 and 90. Such azimuth angles are in the hemisphere for which *u* and *v* are defined.

Data Types: double

el — Elevation angles
[-90:90] (default) | vector of length M

Elevation angles at which pat_azel samples the pattern, specified as a vector of length M. Each elevation angle is in degrees, between –90 and 90.

Data Types: double

Output Arguments

pat_azel — Antenna radiation pattern in azimuth-elevation coordinates
real-valued *M*-by-*L* matrix

Antenna radiation pattern in azimuth-elevation coordinates, returned as a real-valued *M*-by-*L* matrix. pat_azel represents the magnitude pattern. *L* is the length of the az_pat vector, and *M* is the length of the el_pat vector. Units are in dB.

az_pat — Azimuth angles
real-valued length-*L* vector

Azimuth angles at which the `pat_azel` output pattern is sampled, returned as a real-valued length-*L* vector. Units are in degrees.

`el_pat` — Elevation angles
real-valued length-*M* vector

Elevation angles at which the `pat_azel` output pattern is sampled, returned as a real-valued length-*M* vector. Units are in degrees.

More About

U/V Space

The *u* and *v* coordinates are the direction cosines of a vector with respect to the *y*-axis and *z*-axis, respectively.

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles on page 2-490, as follows:

$$u = \sin\theta\cos\phi$$
$$v = \sin\theta\sin\phi$$

In these expressions, φ and θ are the phi and theta angles, respectively.

To convert azimuth and elevation to *u* and *v* use the transformation

$$u = \cos el \sin az$$
$$v = \sin el$$

which is valid only in the range *abs(az)≤=90*.

The values of *u* and *v* satisfy the inequalities

$$-1 \leq u \leq 1$$
$$-1 \leq v \leq 1$$
$$u^2 + v^2 \leq 1$$

Conversely, the phi and theta angles can be written in terms of *u* and *v* using

$$\tan\phi = v/u$$
$$\sin\theta = \sqrt{u^2 + v^2}$$

The azimuth and elevation angles can also be written in terms of *u* and *v*:

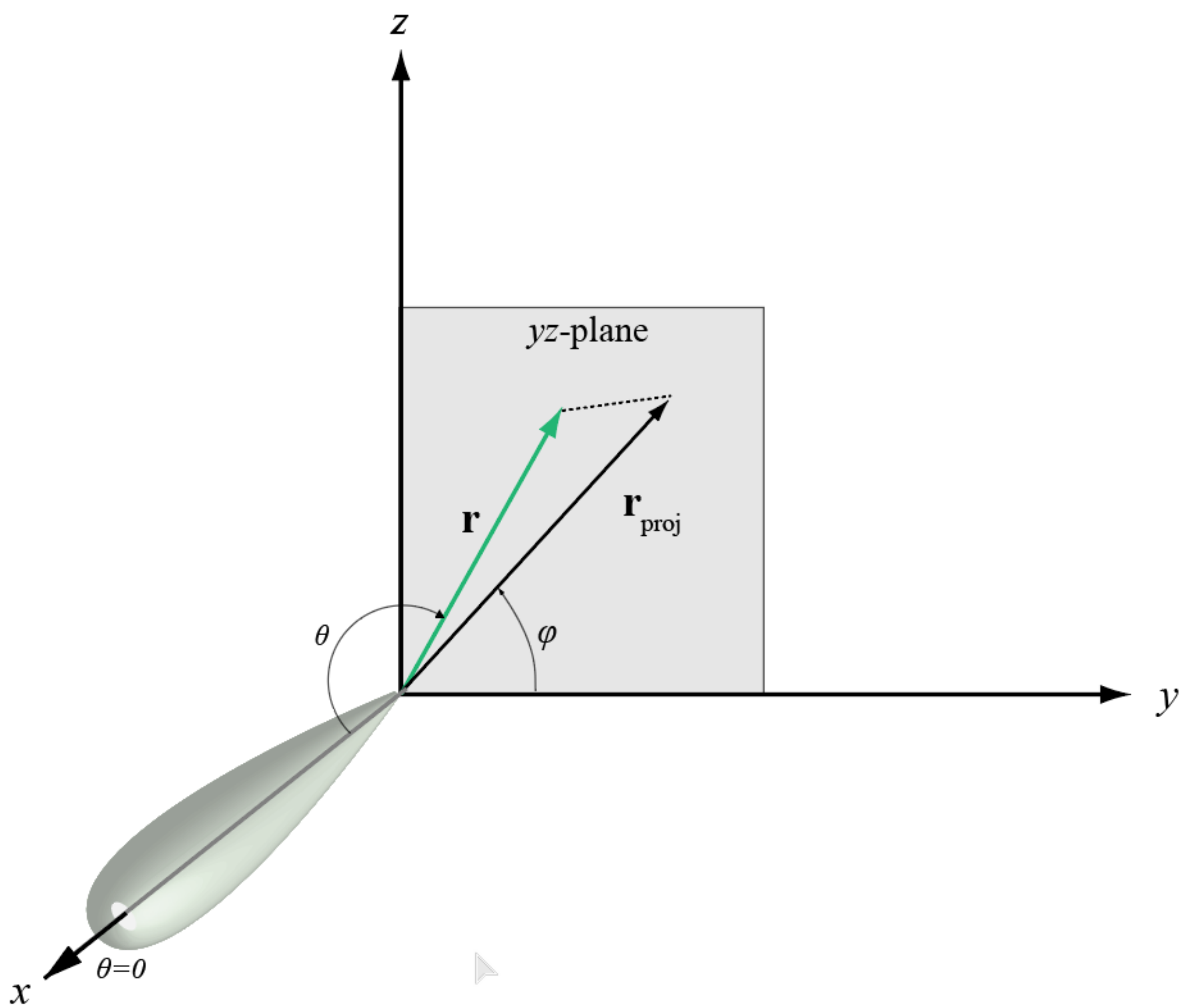
$$\sin el = v$$
$$\tan az = \frac{u}{\sqrt{1 - u^2 - v^2}}$$

Phi Angle, Theta Angle

The phi angle (*φ*) is the angle from the positive *y*-axis to the vector’s orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees.

The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi \sin\theta$

$\tan az = \cos\phi \tan\theta$

$\cos\theta = \cos el \cos az$

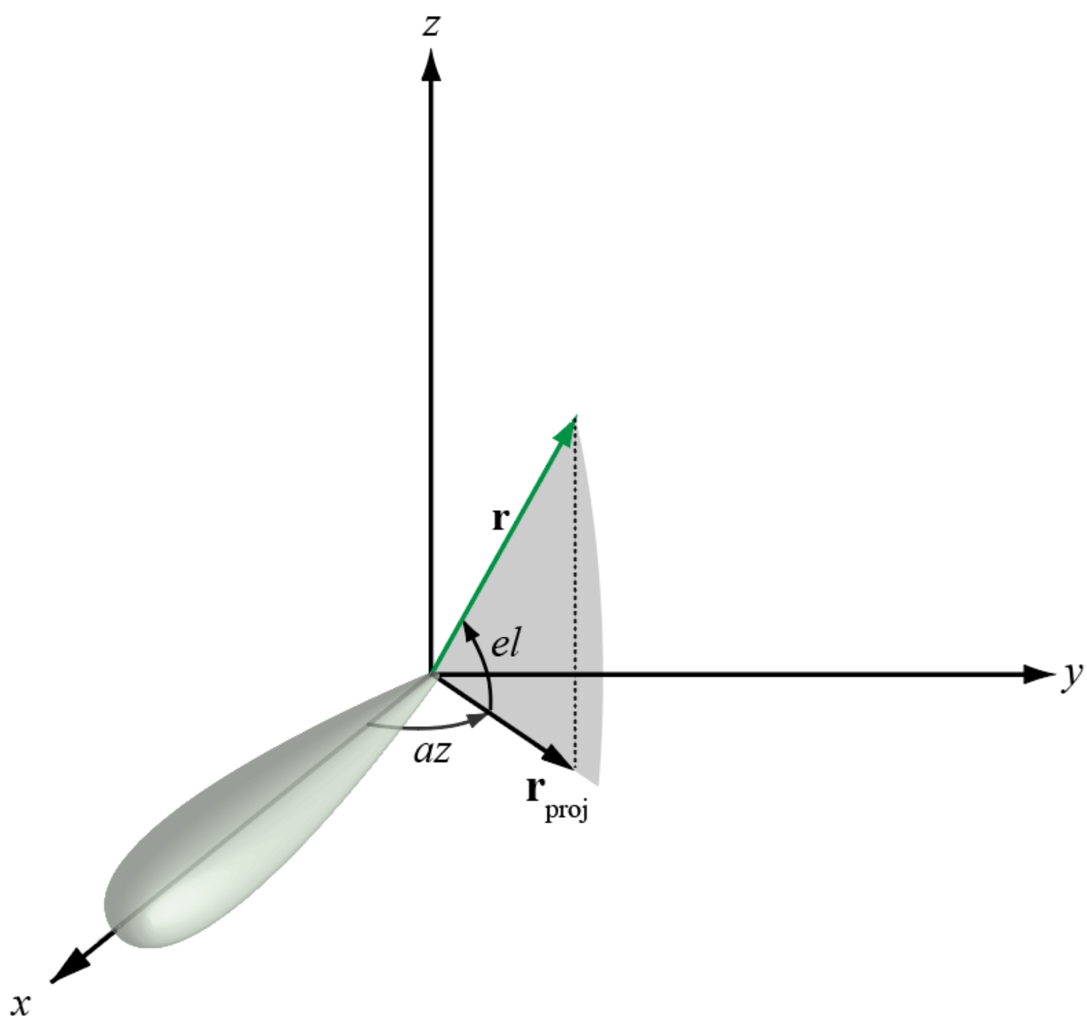
$\tan\phi = \tan el / \sin az$

Azimuth Angle, Elevation Angle

The azimuth angle of a vector is the angle between the *x*-axis and the orthogonal projection of the vector onto the *xy* plane. The angle is positive in going from the *x* axis toward the *y* axis. Azimuth angles lie between –180 and 180 degrees. The elevation angle is the angle between the vector and its orthogonal projection onto the *xy*-plane. The angle is positive when going toward the positive *z*-axis from the *xy* plane. By default, the boresight direction of an element or array is aligned with the positive *x*-axis. The boresight direction is the direction of the main lobe of an element or array.

Note The elevation angle is sometimes defined in the literature as the angle a vector makes with the positive *z*-axis. The MATLAB and Phased Array System Toolbox products do not use this definition.

This figure illustrates the azimuth angle and elevation angle for a vector shown as a green solid line.



Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`phased.CustomAntennaElement` | `uv2azel` | `azel2uv` | `azel2uvpat`

Topics
"Spherical Coordinates"

uv2phitheta

Convert u/v coordinates to phi/theta angles

Syntax

```
PhiTheta = uv2phitheta(UV)
```

Description

`PhiTheta = uv2phitheta(UV)` converts the *u*/*v* space on page 2-495 coordinates to their corresponding phi/theta angle on page 2-495 pairs.

Examples

Conversion of U/V Coordinates

Find the corresponding φ/θ representation for $u = 0.5$ and $v = 0$.

```
PhiTheta = uv2phitheta([0.5; 0])
```

```
PhiTheta = 2×1

         0
   30.0000
```

Input Arguments

UV — Angle in u/v space
two-row matrix

Angle in *u*/*v* space, specified as a two-row matrix. Each column of the matrix represents a pair of coordinates in the form [*u*; *v*]. Each coordinate is between –1 and 1, inclusive. Also, each pair must satisfy $u^2 + v^2 \leq 1$.

Data Types: `double`

Output Arguments

`PhiTheta` — Phi/theta angle pairs
two-row matrix

Phi and theta angles, returned as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [phi; theta]. The matrix dimensions of `PhiTheta` are the same as those of `UV`.

More About

U/V Space

The *u/v* coordinates for the positive hemisphere $x \geq 0$ can be derived from the phi and theta angles on page 2-495.

The relation between the two coordinates is

$$u = \sin\theta\cos\phi$$
$$v = \sin\theta\sin\phi$$

In these expressions, φ and θ are the phi and theta angles, respectively.

To convert azimuth and elevation to *u* and *v* use the transformation

$$u = \cos el \sin az$$
$$v = \sin el$$

which is valid only in the range *abs(az)≤=90*.

The values of *u* and *v* satisfy the inequalities

$$-1 \leq u \leq 1$$
$$-1 \leq v \leq 1$$
$$u^2 + v^2 \leq 1$$

Conversely, the phi and theta angles can be written in terms of *u* and *v* using

$$\tan\phi = v/u$$
$$\sin\theta = \sqrt{u^2 + v^2}$$

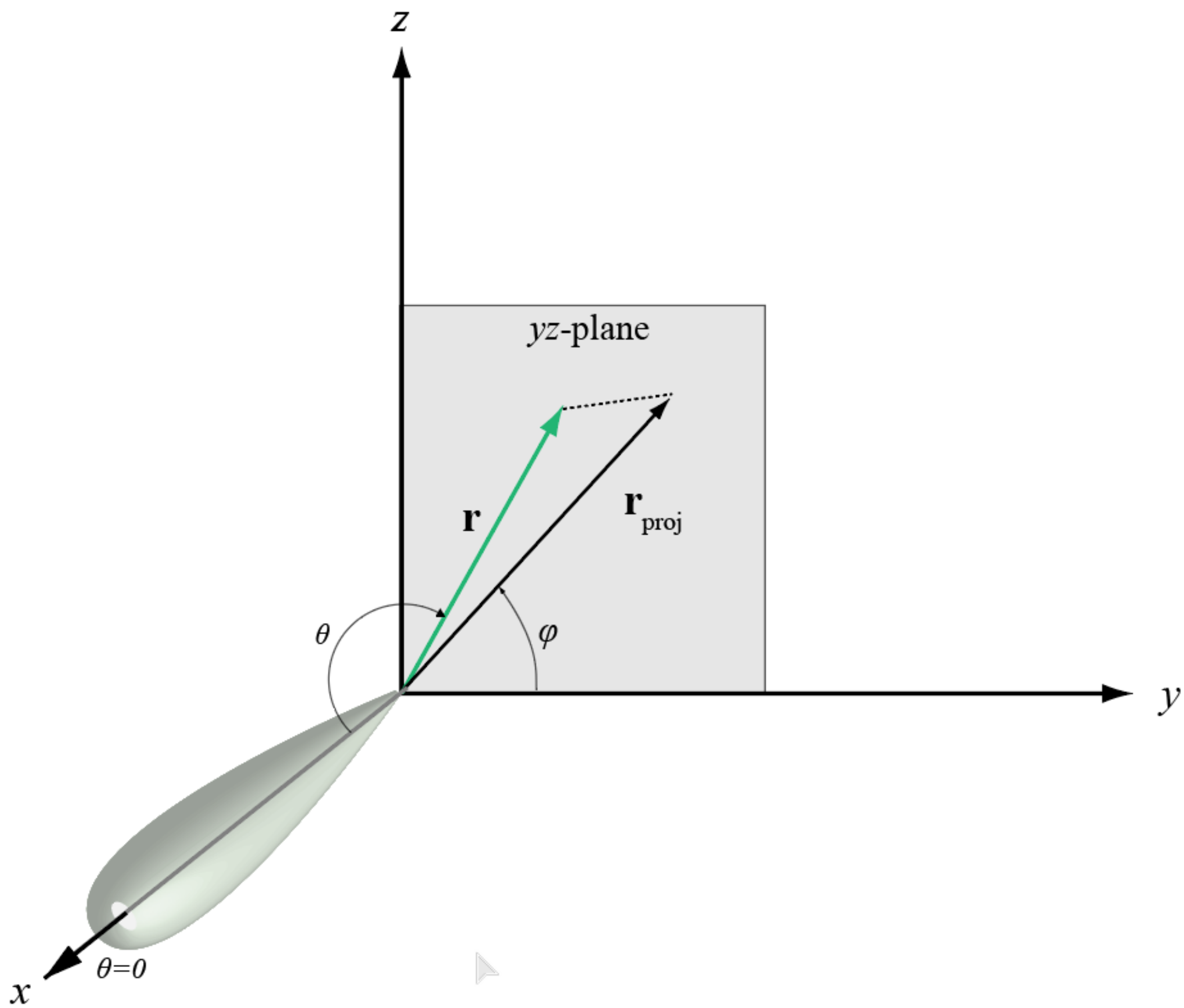
The azimuth and elevation angles can also be written in terms of *u* and *v*:

$$\sin el = v$$
$$\tan az = \frac{u}{\sqrt{1 - u^2 - v^2}}$$

Phi Angle, Theta Angle

The phi angle (φ) is the angle from the positive *y*-axis to the vector's orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi \sin\theta$

$\tan az = \cos\phi \tan\theta$

$\cos\theta = \cos el \cos az$

$\tan\phi = \tan el / \sin az$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`phitheta2uv`

Topics
“Spherical Coordinates”

uv2phithetapat

Convert radiation pattern from u/v form to phi/theta form

Syntax

```
pat_phitheta = uv2phithetapat(pat_uv,u,v)
pat_phitheta = uv2phithetapat(pat_uv,u,v,phi,theta)
[pat_phitheta,phi_pat,theta_pat] = uv2phithetapat( ___ )
```

Description

`pat_phitheta = uv2phithetapat(pat_uv,u,v)` expresses the antenna radiation pattern `pat_phitheta` in φ/θ angle on page 2-502 coordinates instead of u/v space on page 2-502 coordinates. `pat_uv` samples the pattern at *u* angles in `u` and *v* angles in `v`. The `pat_phitheta` matrix uses a default grid that covers φ values from 0 to 360 degrees and θ values from 0 to 90 degrees. In this grid, `pat_phitheta` is uniformly sampled with a step size of 1 for φ and θ. The function interpolates to estimate the response of the antenna at a given direction.

`pat_phitheta = uv2phithetapat(pat_uv,u,v,phi,theta)` uses vectors `phi` and `theta` to specify the grid at which to sample `pat_phitheta`. To avoid interpolation errors, `phi` should cover the range [0, 360], and `theta` should cover the range [0, 90].

`[pat_phitheta,phi_pat,theta_pat] = uv2phithetapat( ___ )` returns vectors containing the φ and θ angles at which `pat_phitheta` samples the pattern, using any of the input arguments in the previous syntaxes.

Examples

Convert Radiation Pattern to φ-θ

Convert a radiation pattern to φ-θ space with the angles spaced 1° apart.

Define the pattern in terms of *u* and *v*. Because *u* and *v* values outside the unit circle are not physical, set the pattern values in this region to zero.

```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Convert the pattern to φ-θ space.

```
[pat_phitheta,phi,theta] = uv2phithetapat(pat_uv,u,v);
```

Plot Converted Radiation Pattern

Convert a radiation pattern to $\phi - \theta$ space with the angles spaced one degree apart.

Define the pattern in terms of u and v. For values outside the unit circle, u and v are undefined, and the pattern value is 0.

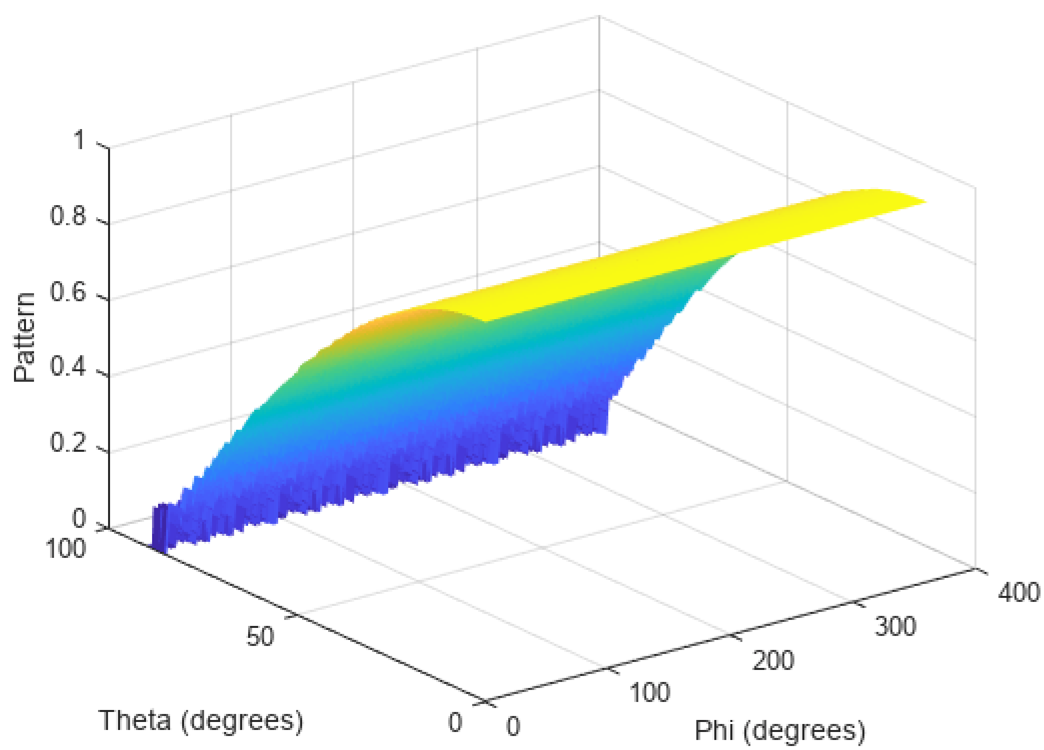
```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Convert the pattern to $\phi - \theta$ space. Store the ϕ and θ angles for use in plotting.

```
[pat_phitheta,phi,theta] = uv2phithetapat(pat_uv,u,v);
```

Plot the result.

```
H = surf(phi,theta,pat_phitheta);
H.LineStyle = 'none';
xlabel('Phi (degrees)');
ylabel('Theta (degrees)');
zlabel('Pattern');
```



Convert Radiation Pattern Using Specific Phi/Theta Values

Convert a radiation pattern to $\phi - \theta$ space with the angles spaced five degrees apart.

Define the pattern in terms of *u* and *v*. For values outside the unit circle, *u* and *v* are undefined, and the pattern value is 0.

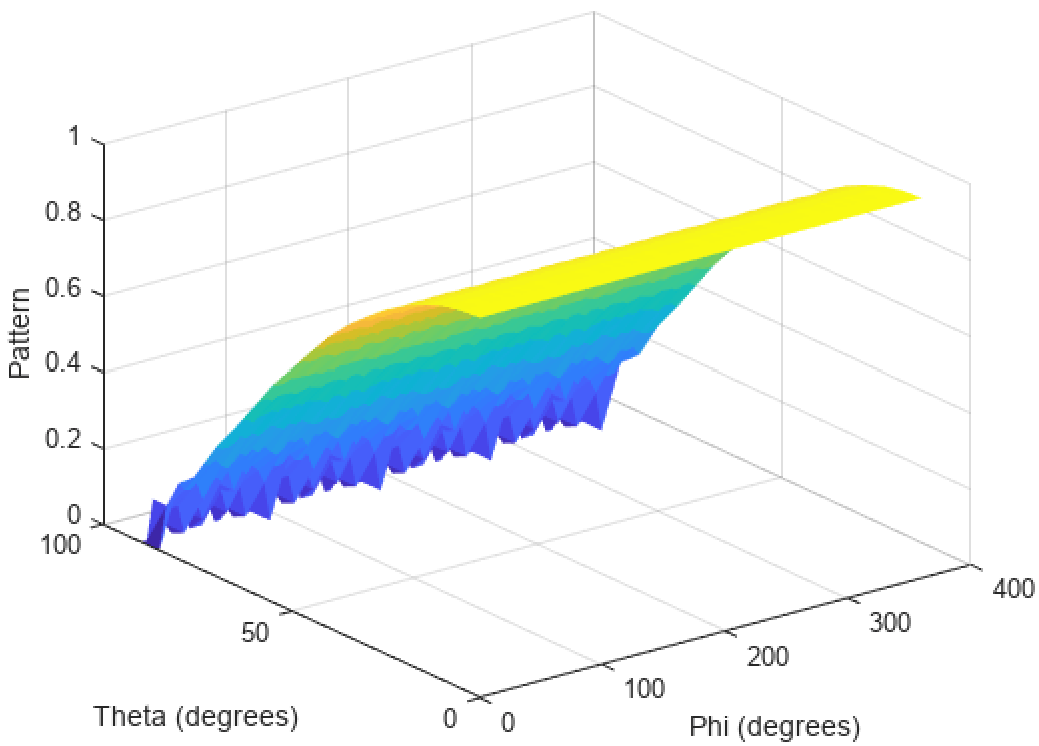
```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Define the set of ϕ and θ angles at which to sample the pattern. Then, convert the pattern.

```
phi = 0:5:360;
theta = 0:5:90;
pat_phitheta = uv2phithetapat(pat_uv,u,v,phi,theta);
```

Plot the result.

```
H = surf(phi,theta,pat_phitheta);
H.LineStyle = 'none';
xlabel('Phi (degrees)');
ylabel('Theta (degrees)');
zlabel('Pattern');
```



Input Arguments

pat_uv — Antenna radiation pattern in *u*/*v* form
Q-by-P matrix

Antenna radiation pattern in *u*/*v* form, specified as a Q-by-P matrix. `pat_uv` samples the 3-D magnitude pattern in decibels, in terms of *u* and *v* coordinates. P is the length of the `u` vector, and Q is the length of the `v` vector.

Data Types: `double`

u — *u* coordinates
vector of length P

u coordinates at which `pat_uv` samples the pattern, specified as a vector of length P. Each coordinate is between –1 and 1.

Data Types: `double`

v — *v* coordinates
vector of length Q

v coordinates at which `pat_uv` samples the pattern, specified as a vector of length Q. Each coordinate is between –1 and 1.

Data Types: `double`

`phi` — Phi angles
`[0:360]` (default) | vector of length L

Phi angles at which `pat_phitheta` samples the pattern, specified as a vector of length L. Each φ angle is in degrees, between 0 and 360.

Data Types: `double`

`theta` — Theta angles
`[0:90]` (default) | vector of length M

Theta angles at which `pat_phitheta` samples the pattern, specified as a vector of length M. Each θ angle is in degrees, between 0 and 90. Such θ angles are in the hemisphere for which *u* and *v* are defined.

Data Types: `double`

Output Arguments

`pat_phitheta` — Antenna radiation pattern in phi-theta coordinates
real-valued *M*-by-*L* matrix

Antenna radiation pattern in phi-theta coordinates, returned as a real-valued *M*-by-*L* matrix. `pat_phitheta` represents the magnitude pattern. *L* is the length of the `phi_pat` vector, and *M* is the length of the `theta_pat` vector. Units are in dB.

`phi_pat` — Phi angles
real-valued length-*L* vector

Phi angles at which the `pat_phitheta` pattern is sampled, returned as a real-valued length *L* vector. Units are in degrees.

`theta_pat` — Theta angles
real-valued length-*M* vector

Theta angles at which the `pat_phitheta` pattern is sampled, returned as a real-valued length-*M* vector. Units are in degrees.

More About

U/V Space

The *u* and *v* coordinates are the direction cosines of a vector with respect to the *y*-axis and *z*-axis, respectively.

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles on page 2-502, as follows:

$$u = \sin\theta\cos\phi$$
$$v = \sin\theta\sin\phi$$

In these expressions, φ and θ are the phi and theta angles, respectively.

To convert azimuth and elevation to *u* and *v* use the transformation

$$u = \cos el \sin az$$
$$v = \sin el$$

which is valid only in the range *abs(az)≤=90*.

The values of *u* and *v* satisfy the inequalities

$$-1 \leq u \leq 1$$
$$-1 \leq v \leq 1$$
$$u^2 + v^2 \leq 1$$

Conversely, the phi and theta angles can be written in terms of *u* and *v* using

$$\tan\phi = v/u$$
$$\sin\theta = \sqrt{u^2 + v^2}$$

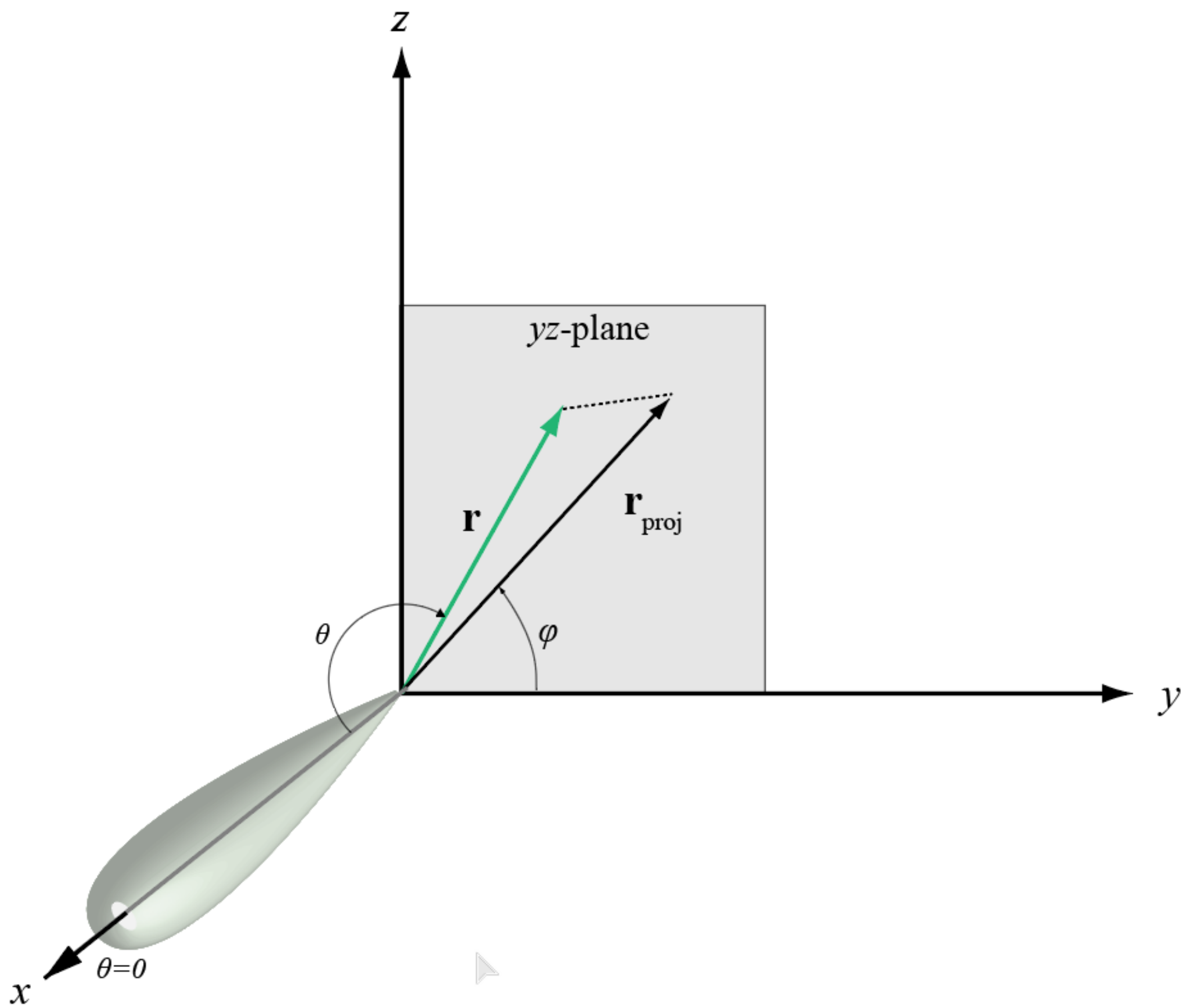
The azimuth and elevation angles can also be written in terms of *u* and *v*:

$$\sin el = v$$
$$\tan az = \frac{u}{\sqrt{1 - u^2 - v^2}}$$

Phi Angle, Theta Angle

The phi angle (φ) is the angle from the positive *y*-axis to the vector’s orthogonal projection onto the *yz* plane. The angle is positive toward the positive *z*-axis. The phi angle is between 0 and 360 degrees. The theta angle (θ) is the angle from the *x*-axis to the vector itself. The angle is positive toward the *yz* plane. The theta angle is between 0 and 180 degrees.

The figure illustrates phi and theta for a vector that appears as a green solid line.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$\sin el = \sin\phi \sin\theta$

$\tan az = \cos\phi \tan\theta$

$\cos\theta = \cos el \cos az$

$\tan\phi = \tan el / \sin az$

Version History

Introduced in R2012a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also
`phased.CustomAntennaElement` | `uv2phitheta` | `phitheta2uv` | `phitheta2uvpat`

Topics
"Spherical Coordinates"

val2ind

Uniform grid index

Syntax

```
indx = val2ind(val,delta)
indx = val2ind(val,delta,startval)
```

Description

`indx = val2ind(val,delta)` returns the index of the value `val` in a uniform grid with a spacing between elements of `delta`. The first element of the uniform grid is zero. If `val` does not correspond exactly to an element of the grid, the next element is returned. If `val` is a vector, then `indx` is a vector of the same size.

`indx = val2ind(val,delta,startval)` specifies the starting value of the uniform grid as `startval`.

Examples

Compute Index of Value in Grid

Find the index corresponding to 0.0001 in a uniform grid with 1 MHz sampling rate.

```
fs = 1e6;
indx = val2ind(0.0001,1/fs)
```

```
indx = 101
```

Compute Indices of Values in Grid

Find the indices corresponding to a vector of values in a uniform grid with 1 kHz sampling rate. Values are not divisible by 1/fs.

```
fs = 1.0e3;
values =[0.0095 0.0125 0.0225];
indx = val2ind(values,1/fs)
```

```
indx = 1×3

    11    14    24
```

Input Arguments

`val` — Data values
scalar | vector

Data values, specified as a scalar or vector.

Example: [3.4 6.3 9.8 12.1]

Data Types: double

delta — Spacing between grid elements
positive scalar

Spacing between grid elements, specified as a positive scalar.

Example: 1/1e3

Data Types: double

startval — Starting value of the uniform grid
0 (default) | real scalar

Starting value of the uniform grid, specified as a scalar startval must be smaller than or equal to the minimum value in val.

Data Types: double

Version History

Introduced in R2011a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Does not support variable-size inputs.

See Also

find | sub2ind | unigrid

Topics

"Array Indexing"

waterfill

Waterfill MIMO power distribution

Syntax

```
P = waterfill(Pt,Pn)
waterfill(Pt,Pn)
```

Description

`P = waterfill(Pt,Pn)` optimally distributes the total transmitted power, `Pt`, among multiple channels to maximize channel capacity. The argument `Pn` represents the noise in each channel. The function can optimize independent subcarriers simultaneously.

`waterfill(Pt,Pn)` displays a waterfill diagram.

Examples

Compute Distributed Power Using Waterfill

Using the waterfill algorithm, compute the distributed power per channel for two subcarriers. There are four channels per subcarrier.

Specify the same total power for both subcarriers using a scalar value.

```
Pt = 10;
```

Specify the noise power. The rows correspond to the subcarriers and the columns to the channels.

```
Pn = [1 4 6 3; 5 4 3 6];
P = waterfill(Pt,Pn)
```

```
P = 2×4

     5     2     0     3
     2     3     4     1
```

Now, specify a different total power for each subcarrier.

```
Pt = [10,5];
P = waterfill(Pt,Pn)
```

```
P = 2×4

    5.0000    2.0000         0    3.0000
    0.6667    1.6667    2.6667         0
```

Plot Distributed Power Using Waterfill

Using the waterfill algorithm, plot the distributed power per channel for two subcarriers. There are four channels per subcarrier.

Specify a different total power for each subcarrier.

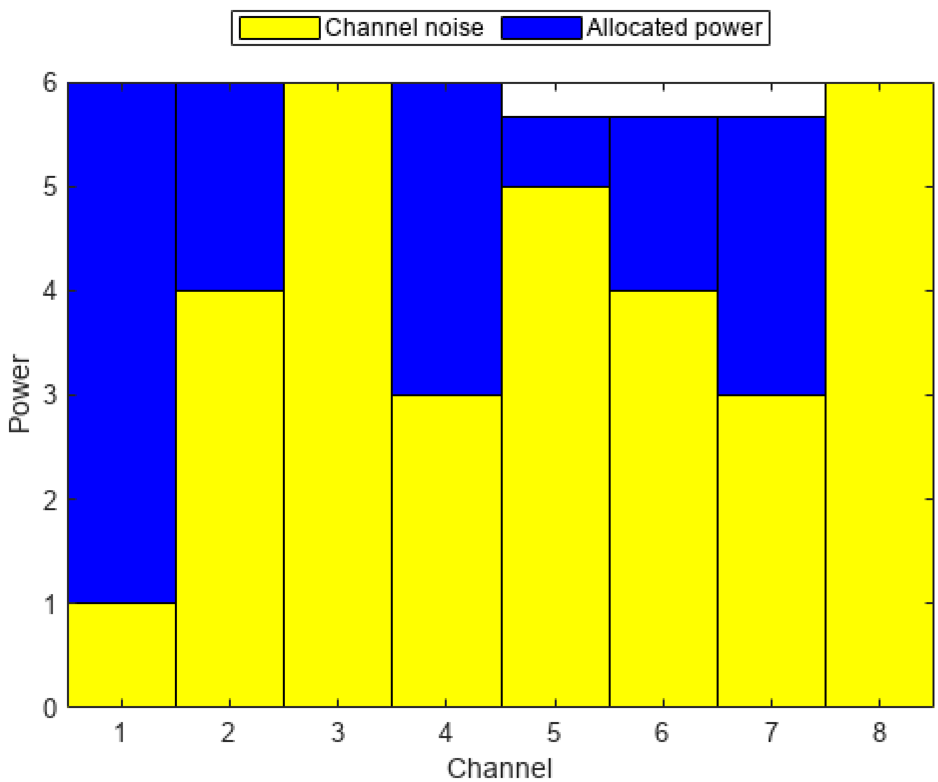
```
Pt = [10,5];
```

Specify the noise power. The rows correspond to the subcarriers and the columns to the channels.

```
Pn = [1 4 6 3; 5 4 3 6];
```

Display the waterfill plot.

```
waterfill(Pt,Pn)
```



Input Arguments

Pt — Total transmitted power
positive scalar | positive-valued *L*-element row or column vector

Total transmitted power per subcarrier, specified as a positive-valued *L*-element row or column vector where *L* is the number of subcarriers. When `Pt` is a scalar, all subcarriers have the same power. When `Pt` is a vector, the total power in a subcarriers is given by the corresponding element in `Pt`. Units are arbitrary.

Example: `[20 30]`

Data Types: `double`

Pn — Channel noise power
positive-valued *N*-element row or column vector | positive-valued *L*-by-*N*-element matrix

Channel noise powers, specified as a positive-valued *N*-element row or column vector or a positive-valued *L*-by-*N*-element matrix. *N* is the number of channels and *L* is the number of subcarriers. If `Pn` is a vector, each element represents the noise power in the corresponding channel. The noise powers for each channel is the same for all subcarriers. If `Pn` is a matrix, an element in the matrix represents the noise power in the corresponding channel at the corresponding subcarrier. Units are arbitrary but must match the units for `Pt`.

Example: `[10 20 15]`

Data Types: `double`

Output Arguments

P — Allocated power per channel
positive-valued *L*-by-*N*-element matrix

Allocated power per channel, specified as a positive-valued *L*-by-*N*-element matrix. *N* is the number of channels and *L* is the number of subcarriers. Units are the same as the transmitted power, `Pt`. Each row corresponds to a subcarrier and specifies the distributed power for the channels in the subcarrier. Units are the same as for `Pt` and `Pn`.

Data Types: `double`

Algorithms

The number of subcarriers is determined by either the dimensions of `Pt` or `Pn`.

- When you specify `Pt` as an *L*-element vector, there are *L* subcarriers with different total powers. If you specify `Pn` as *N*-element vector, this noise power vector is the same for all subcarriers. If you specify `Pn` as an *L*-by-*N* matrix, each row applies to the corresponding subcarrier.
- When you specify `Pt` as a scalar, `Pn` determines the number of subcarriers. If you specify `Pn` as an *N*-element vector, each element is the noise power in a channel and there is only one subcarrier. If you specify `Pn` as an *L*-by-*N* matrix, there are *L* subcarriers all having the same transmitted power.

Version History

Introduced in R2017a

References

[1] Heath, R. Jr. et al. "An Overview of Signal Processing Techniques for Millimeter Wave MIMO Systems", arXiv.org:1512.03007 [cs.IT], 2015.

[2] Tse, D. and P. Viswanath, *Fundamentals of Wireless Communications*, Cambridge: Cambridge University Press, 2005.

[3] Paulraj, A. *Introduction to Space-Time Wireless Communications*, Cambridge: Cambridge University Press, 2003.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Does not support variable-size inputs.
- Supported only when output arguments are specified.

See Also

Functions
`diagbfweights` | `scatteringchanmtx`

Objects
`phased.ScatteringMIMOChannel`

tirempl

Path loss using Terrain Integrated Rough Earth Model (TIREM)

Syntax

```
pl = tirempl(r,z,f)
pl = tirempl(r,z,f,Name,Value)
[pl,output] = tirempl( ___ )
```

Description

`pl = tirempl(r,z,f)` returns the path loss in dB for a signal with frequency `f` when it is propagated over terrain. You can specify terrain using numeric vectors for distance `r` and elevation `z` along the great circle path between the transmitter and the receiver. The Terrain Integrated Rough Earth Model™ (TIREM™) model combines physics with empirical data to provide path loss estimates. The TIREM model is valid from 1 MHz to 1000 GHz.

Note `tirempl` requires access to the external TIREM library. Use `tiremSetup` to set up access.

`pl = tirempl(r,z,f,Name,Value)` returns the path loss in dB with additional options specified by name-value pairs.

`[pl,output] = tirempl( ___ )` returns the path loss, `pl`, and the output structure containing the information on the TIREM analysis.

Examples

Path Loss Over Flat Terrain

Calculate the path loss over flat terrain. Define the terrain profile for distances up to 10 km with step size of 100 m.

```
freq = 28e9;
r = 0:100:10000;
z = zeros(1,numel(r));
    Lterrain1 = tirempl(r,z,freq,...
       'TransmitterAntennaHeight',5, ...
       'ReceiverAntennaHeight',5)
```

```
Lterrain1 =

  142.6089
```

Input Arguments

r — Distances
numeric vector

Distances along the great circle path between the transmitter and the receiver, specified as a numeric vector with each value in meters. The number of distance values must be equal to the number of elevation values.

Data Types: `double`

`z` — Elevation
numeric vector

Elevation values corresponding to the distance values along the great circle path between the transmitter and the receiver, specified as a numeric vector with each value in meters. The number of elevation values must be equal to the number of distance values.

Data Types: `double`

`f` — Frequency of propagated signal
scalar | numeric vector

Frequency of the propagated signal, specified as a scalar or numeric vector with each element unit in Hz.

Data Types: `double`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'TransmitterAntennaHeight',50`

`TransmitterAntennaHeight` — Transmitter antenna height above ground
`10` (default) | numeric scalar

Transmitter antenna height above the ground, specified as a numeric scalar in the range of 0 to 30000. The height is measured from ground elevation to the center of the antenna.

Data Types: `double`

`ReceiverAntennaHeight` — Receiver antenna height above ground
`1` (default) | numeric scalar

Receiver antenna height above the ground, specified as a numeric scalar in the range of 0 to 30000. The height is measured from ground elevation to the center of the antenna.

Data Types: `double`

`AntennaPolarization` — Polarization of transmitter and receiver antennas
`'horizontal'` (default) | `'vertical'`

Polarization of the transmitter and the receiver antennas, specified as `'horizontal'` or `'vertical'`.

Data Types: `string` | `char`

`GroundConductivity` — Conductivity of ground
`0.005` (default) | numeric scalar

Conductivity of the ground, specified as a numeric scalar in the range of 0.00005 to 100 in Siemens per meter. This value is used to calculate the path loss due to ground reflection. The default value corresponds to the average ground conductivity.

Data Types: `double`

`GroundPermittivity` — Relative permittivity of ground

`15` (default) | numeric scalar

Relative permittivity of the ground, specified as a numeric scalar in the range of 1 to100. Relative permittivity is the ratio of absolute material permittivity to the permittivity of vacuum. This value is used to calculate the path loss due to ground reflection. The default value corresponds to the average ground permittivity.

Data Types: `double`

`AtmosphericRefractivity` — Atmospheric refractivity near ground

`301` (default) | numeric scalar

Atmospheric refractivity near the ground, specified as a numeric scalar in N-units in the range of 250 to 400. This value is used to calculate the path loss due to atmospheric refraction and tropospheric scatter. The default value corresponds to average atmospheric conditions.

Data Types: `double`

`Humidity` — Absolute air humidity near ground

`9` (default) | numeric scalar

Absolute air humidity near the ground, specified as a numeric scalar in g/m^^{3} in the range of 50 to 110. This value is used to calculate path loss due to atmospheric absorption. The default value corresponds to the absolute humidity of air at 15 degrees Celsius and 70 percent relative humidity.

Data Types: `double`

Output Arguments

`pl` — Path loss

scalar | 1-by-*N* vector

Path loss, returned as a scalar or 1-by-*N* vector with each element unit in decibels. *N* is the number of frequencies defined in the input `f`.

Path loss is calculated from free-space loss, terrain diffraction, ground reflection, refraction through the atmosphere, tropospheric scatter, and atmospheric absorption.

`output` — Information of TIREM analysis

structure

Information of TIREM analysis, returned as a structure. Each field of the structure represents an output from TIREM analysis.

Version History

Introduced in R2019b

See Also
`tiremSetup`

Topics
"Access TIREM Software"

tiremSetup

Set up access to Terrain Integrated Rough Earth Model (TIREM)

Syntax

```
tiremSetup
tiremSetup(libfolder)
libfolder = tiremSetup
```

Description

`tiremSetup` opens a dialog to select the Terrain Integrated Rough Earth Model (TIREM) library folder. The TIREM library folder must contain the `tirem3` shared library, where the full library name is platform dependent. For more information, see .“Platform dependent library names” on page 2-515

`tiremSetup(libfolder)` sets the TIREM library folder to `libfolder`.

`libfolder = tiremSetup` returns the current TIREM library folder.

Input Arguments

`libfolder` — Name of TIREM library folder
character vector

Name of the TIREM library folder, specified as a character vector.

Data Types: `char` | `string`

Output Arguments

`libfolder` — Current TIREM library folder
character vector | string scalar

Current TIREM library folder, returned as a character vector or a string scalar. If TIREM access has not been setup, `libfolder` is empty.

More About

Platform dependent library names

Platform	Shared library name
`Windows`	`libtirem3.dll` or `tirem3.dll`
`Linux`	`libtirem3.so`
`Mac`	`libtirem3.dylib`

Version History

Introduced in R2019b

See Also

`tirempl`

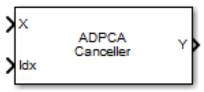
Topics

"Access TIREM Software"

Blocks

ADPCA Canceller

Adaptive displaced phase center array (ADPCA) pulse canceller for a uniform linear array



Libraries:
Phased Array System Toolbox / Space-Time Adaptive Processing

Description

The ADPCA Canceller block filters clutter impinging on a uniform linear array using a displaced phase center array pulse canceller.

Ports

Input

X — Input signal
M-by-*N*-by-*P* complex-valued matrix

Input signal, specified as an *M*-by-*N*-by-*P* complex-valued array. *M* is the number of range samples, *N* is the number of channels, and *P* is the number of pulses.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Idx — Index of range cells
positive integer

Index of range cells to compute processing weights.

Example: `1`

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, specified as a positive scalar.

Dependencies

To enable this port, set the **Specify PRF as** parameter to `Input port`.

Data Types: `double`

Ang — Targeting direction
2-by-1 real-valued vector

Targeting direction, specified as a 2-by-*1* real-valued vector. The vector takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie

between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Specify direction as** parameter to `Input port`.

Data Types: `double`

Dop — Targeting Doppler frequency
scalar

Targeting Doppler frequency of current pulse, specified as a scalar.

Dependencies

This port appears when the **Output pre-Doppler result** check box is cleared and the **Specify targeting Doppler as** parameter is set to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-1 complex-valued vector

Processing output, returned as an *M*-by-*1* complex-valued vector. The quantity *M* is the number of range samples in the input port X.

Data Types: `double`

W — Processing weights
length *N**P* complex-valued vector

Processing weights, returned as Length *N**P* complex-valued vector. The quantity *N* is the number of channels and *P* is the number of pulses. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays. *L* is the number of desired beamforming directions specified in the Ang input port or by the **Beamforming direction (deg)** parameter. There is one set of weights for each beamforming direction.

Dependencies

To enable this port, select the **Enable weights output** check box.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Specify PRF as — Source of PRF value
`Property` (default) | `Input port`

Source of PRF value, specified as `Property` or `Input port`. When set to `Property`, the **Pulse repetition frequency (Hz)** parameter sets the PRF. When set to `Input port`, pass in the PRF using the PRF input port.

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1` (default) | positive scalar

Pulse repetition frequency, PRF, specified as a positive scalar. Units are in Hertz. Set this parameter to the same value set in any `Waveform` library block used in the simulation.

Dependencies

To enable this parameter, set the **Specify PRF as** parameter to `Property`.

Specify direction as — Specify source of targeting directions
`Property` (default) | `Input port`

Specify whether the targeting direction for the STAP processor block comes from a block parameter or from the ANG input port. Values of this parameter are

`Property`	• For the ADPCA Canceller and DPCA Canceller blocks, targeting direction is specified using **Receiving mainlobe direction (deg)**. • For the SMI Beamformer block, targeting direction is specified using **Targeting direction**. These parameters appear only when the **Specify direction as** parameter is set to `Property`.
`Input port`	Enter the targeting directions using the Ang input port. This port appears only when **Specify direction as** is set to `Input port`.

Receiving mainlobe direction (deg) — Pointing direction of main lobe of array
`[0;0]` (default) | real-valued 2-by-1 vector

Specify the direction of the main lobe of the receiving sensor array as a real-valued 2-by-1 vector. The direction is specified in the format of `[AzimuthAngle; ElevationAngle]`. The azimuth angle should be between –180° and 180° and the elevation angle should be between –90° and 90°.

Example: `[100;-45]`

Dependencies

To enable this parameter, set **Specify direction as** to `Property`.

Number of bits in phase shifters — Number of phase shift quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Specify targeting Doppler as — Source of targeting Doppler
`Property` (default) | `Input port`

Specify whether targeting Doppler values for the STAP processor comes from the **Targeting Doppler (Hz)** parameter of this block or using the `DOP` input port. For the ADPCA Canceller and DPCA Canceller blocks, the **Specify targeting Doppler as** parameter appears only when the **Output pre-Doppler result** check box is cleared. Values of this parameter are

`Property`	Specify targeting Doppler values using the **Targeting Doppler** parameter of the block. The **Targeting Doppler** parameter appears only when **Specify targeting Doppler as** is set to `Property`.
`Input port`	Specify targeting Doppler values using the `Dop` input port. This port appears only when **Specify targeting Doppler as** is set to `Input port`.

Targeting Doppler (Hz) — Targeting Doppler of STAP processor
`0` (default) | scalar

Targeting Doppler of STAP processor, specified as a scalar.

Dependencies

- To enable this parameter for the SMI Beamformer block, set **Specify targeting Doppler as** to `Property`.
- To enable this parameter for the ADPCA Canceller and DPCA Canceller blocks, first clear the **Output pre-Doppler result** check box. Then set the **Specify targeting Doppler as** parameter to `Property`.

Number of guard cells — Number of guard cells using for training
2 (default) | positive even integer

Number of guard cells used for training, specified as a positive, even integer. Whenever possible, the set of guard cells is equally divided into regions before and after the test cell.

Number of training cells — Number of cells used for training
2 (default) | positive even integer

Number of cells used for training, specified as a positive even integer. Whenever possible, the set of training cells is equally divided into regions before and after the test cell.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Output pre-Doppler result — Output results before Doppler filtering
on (default) | off

Select this check box to output the results before Doppler filtering. Clear this check box to output the processing result after Doppler filtering. Selecting this check box will remove the **Specify targeting Doppler as** and **Targeting Doppler (Hz)** parameters.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum

Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
on (default) | off

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees

azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Specify sensor array as — Type of array
`Array (no subarrays)` (default) | `MATLAB expression`

Specify a ULA sensor array directly or by using a MATLAB expression.

Types

`Array (no subarrays)`
`MATLAB expression`

Number of elements — Number of array elements in U
2 (default) | positive integer greater than or equal to two

The number of array elements for ULA arrays, specified as an integer greater than or equal to two.

Example: `11`

Data Types: `double`

Element spacing — Distance between ULA elements
`0.5` (default) | positive scalar

Distance between adjacent ULA elements, specified as a positive scalar. Units are in meters.

Example: `1.5`

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Taper — ULA array taper
`1` (default) | complex-valued vector

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Example: [0.5;1;0.5]

Data Types: `double`

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create a ULA array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.ULA('NumElements',13)`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Version History

Introduced in R2014b

See Also

`phased.ADPCACanceller`

Angle Doppler Response

Angle-Doppler response

Libraries:
Phased Array System Toolbox / Space-Time Adaptive Processing

Description

The Angle Doppler Response block computes the angle-Doppler response of the input signal. The output response is a matrix whose rows represent Doppler bins and whose columns represent angle bins.

Ports

Input

X — Input data
M-by-*N* complex-valued matrix | *M***N*-element complex-valued vector

Input signal, specified as an *M*-by-*N* complex-valued matrix or an *M***N* complex-valued vector. *M* is the number of array elements or the number of subarrays, if the array supports subarrays, specified in the **Sensor Array** panel. *N* is the number of data samples. *N* must be greater than or equal to two.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, specified as a positive scalar.

Dependencies

To enable this port, set the **Specify PRF as** parameter to `Input port`.

Data Types: `double`

El — Elevation angle
scalar

Elevation angle, specified as a scalar. Angle units are in degrees. The elevation angle must lie between –90° and 90°, inclusive.

Dependencies

To enable this port, set **Source of elevation angle** to `Input port`.

Data Types: `double`

Output

Resp — Angle-Doppler response
P-by-*Q* complex-valued matrix

Angle Doppler response, returned as a *P*-by-*Q* matrix. *P* is specified by the **Number of Doppler bins** parameter and *Q* is specified by the **Number of angle bins** parameter.

Data Types: `double`

Ang — Response-matrix angle values
Q-by-1 real-valued vector

Response-matrix angle values, returned as a *Q*-by-1 real-valued vector. The angle values correspond to the columns of the angle-Doppler response matrix. *Q* is specified by the **Number of angle bins** parameter.

Data Types: `double`

Dop — Response-matrix Doppler values
P-by-1 real-valued vector

Response-matrix Doppler values, returned as a *P*-by-1 real-valued vector. The Doppler values correspond to the rows of the angle-Doppler response matrix. *P* is specified by the **Number of Doppler bins** parameter.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Specify PRF as — Source of PRF value
`Property` (default) | `Input port`

Source of PRF value, specified as `Property` or `Input port`. When specifier as `Property`, the **Pulse repetition frequency (Hz)** parameter sets the PRF. When set to `Input port`, pass in the PRF using the PRF input port.

Pulse repetition frequency (Hz) — Pulse repetition frequency
1 (default) | positive scalar

Pulse repetition frequency, PRF, specified as a positive scalar. Units are in Hertz. Set this parameter to the same value set in any Waveform library block used in the simulation.

Dependencies

To enable this parameter, set the **Specify PRF as** parameter to Property.

Source of elevation angle — Elevation angle source
Property (default) | Input port

Elevation angle source, specified as Property or Input port. Values of this parameter are

Property	The **Elevation angle (deg)** parameter of this block specifies the elevation angle.
Input port	The elevation angle is set using the El input port.

Elevation angle (deg) — Elevation angle used to calculate the angle-Doppler response
0 (default) | scalar

Elevation angle used to calculate the angle-Doppler response, specified as a scalar. Units are degrees. The angle must be between –90° and 90°.

Example: -45

Dependencies

To enable this parameter, set **Source of elevation angle** to Property

Data Types: double

Number of angle bins — Number of angle samples
256 (default) | positive integer greater than two

The number of samples in the angular domain used to calculate the angle-Doppler response, specified as a positive integer greater than two.

Example: 600

Data Types: double

Number of Doppler bins — Number of angle samples
256 (default) | positive integer greater than two

The number of samples in the Doppler domain used to calculate the angle-Doppler response, specified as a positive integer greater than two.

Example: 128

Data Types: double

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><td rowspan="2">Block Simulation</td><td colspan="3">Simulation Behavior</td></tr>
<tr><td><code>Normal</code></td><td><code>Accelerator</code></td><td><code>Rapid Accelerator</code></td></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-L row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
ULA (default) | URA | UCA | `Conformal Array`

Array geometry, specified as one of

- ULA — Uniform linear array
- URA — Uniform rectangular array
- UCA — Uniform circular array
- Conformal Array — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or UCA.

Element spacing (m) — Spacing between array elements
0.5 for ULA arrays and [0.5,0.5] for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form [SpacingBetweenArrayRows,SpacingBetweenArrayColumns].
- When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or URA.

Array axis — Linear axis direction of ULA
y (default) | x | z

Linear axis direction of ULA, specified as y, x, or z. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to ULA.
- This parameter is also enabled when the block only supports ULA arrays.

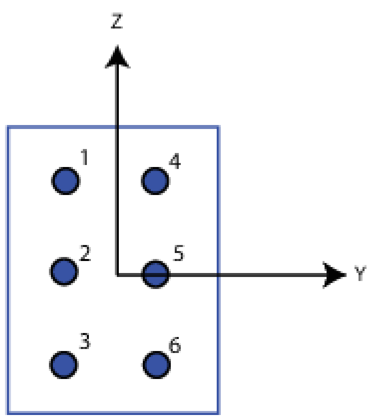
Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

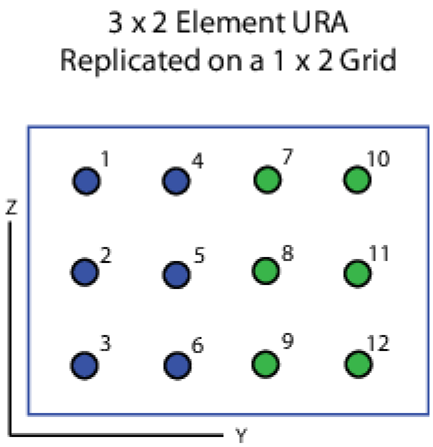
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as ULA or URA.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

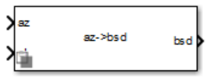
Introduced in R2014b

See Also

`phased.AngleDopplerResponse`

Azimuth Broadside Converter

Convert azimuth angle to broadside angle or broadside angle to azimuth angle



Libraries:
Phased Array System Toolbox / Environment and Target

Description

The Azimuth Broadside Converter block converts an angle direction expressed in terms of “Broadside Angles” into the equivalent azimuth angle or converts from azimuth angle into the equivalent broadside angle. In both cases, you must specify the elevation angle.

Ports

Input

az — Azimuth angle
scalar | vector of real-values

Azimuth angle of direction, specified as a scalar or vector of real-values. Units are in degrees. When `az` is a vector, the dimensions of `az` and `el` must match.

Dependencies

To enable this port, set **Conversion Mode** to **azimuth -> broadside**.

Data Types: `double`

bsd — Broadside angle
scalar | vector of real-values

Broadside angle of direction, specified as a scalar or vector of real-values. Units are in degrees. When `bsd` is a vector, the dimensions of `bsd` and `el` must match.

Dependencies

To enable this port, set **Conversion Mode** to **broadside -> azimuth**.

Data Types: `double`

el — Elevation angle
scalar | vector of real-values

Elevation angle of direction, specified as a scalar or vector of real-values. Units are in degrees. The dimensions of `el` must match the dimensions of `az` and `bsd`.

Data Types: `double`

Output

az — Azimuth angle
scalar | vector of real-values

Azimuth angle of direction, returned as a scalar or vector of real-values. Units are in degrees.

Dependencies

To enable this port, set **Conversion Mode** to **broadside -> azimuth**.

Data Types: `double`

bsd — Broadside angle
scalar | vector of real-values

Broadside angle of direction, returned as a scalar or vector of real-values. Units are in degrees.

Dependencies

To enable this port, set **Conversion Mode** to **azimuth -> broadside**.

Data Types: `double`

Parameters

Conversion mode — Angle conversion type
broadside -> azimuth (default) | **azimuth -> broadside**

Angle conversion type, specified as

broadside -> azimuth	Convert direction expressed in broadside and elevation angles to azimuth and elevation angles.
azimuth -> broadside	Convert direction expressed in azimuth and elevation angles to broadside and elevation angles.

Version History

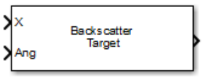
Introduced in R2014b

See Also

`az2broadside` | `broadside2az`

Backscatter Radar Target

Backscatter radar target



Libraries:
Phased Array System Toolbox / Environment and Target

Description

The Backscatter Radar Target block models the monostatic radar reflections of nonpolarized electromagnetic signals from a target. Target model includes all four Swerling target fluctuation models and non-fluctuating model. You can model several targets simultaneously by specifying multiple radar cross-section (RCS) matrices.

Ports

Input

X — Narrowband signal
N-by-*1* complex-valued vector | *N*-by-*M* complex-valued matrix

Narrowband nonpolarized signal, specified as an *N*-by-*1* complex-valued vector or an *N*-by-*M* complex-valued matrix. The quantity *N* is the number of signal samples and *M* is the number of signals reflecting from the target. Each column corresponds to an independent signal incident at a different reflecting angle.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Ang — Incident signal direction
2-by-1 real-valued column vector | 2-by-*M* real-valued column matrix

Incident signal direction, specified as a 2-by-1 real-valued column vector or a 2-by-*M* real-valued column matrix. *M* is the number of signals reflecting from the target. Each column of `Ang` specifies the incident direction of the corresponding signal in the form of an `[AzimuthAngle;ElevationAngle]` pair. Units are degrees. The number of columns in `Ang` must match the number of independent signals in `X`.

Example: `[30;45]`

Data Types: `double`

Update — Switch to update RCS
`false` | `true`

Switch to update RCS fluctuation model values, specified as `false` or `true`. When `Update` is `true`, the RCS value is updated. If `Update` is `false`, the RCS remains unchanged.

Dependencies

To enable this port, set the **Fluctuation model** drop-down menu to `Swerling1`, `Swerling2`, `Swerling3`, or `Swerling4`.

Output

Port_1 — Narrowband reflected signal
1-by-*M* complex-valued vector | *N*-by-*M* complex-valued matrix

Narrowband nonpolarized signal, specified as an 1-by-*M* complex-valued vector or a *N*-by-*M* complex-valued matrix. Each column contains an independent signal reflected from the target.

The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to a different reflecting angle.

The output port contains signal samples arriving at the signal destination within the current input time frame. When the propagation time from source to destination exceeds the current time frame duration, the output does not contain all contributions from the input of the current time frame.

Parameters

Azimuth angles (deg) — Azimuth angles

`[-180:180]` (default) | 1-by-*P* real-valued row vector | *P*-by-1 real-valued column vector

Specify the azimuth angles used to define the angular coordinates of the **RCS pattern (m^2)** parameter. Specify azimuth angles as a length *P* vector. Units are degrees. *P* must be greater than two. This parameter determines the incident azimuthal arrival angle of any element of the cross-section patterns.

Data Types: `double`

Elevation angles (deg) — Elevation angles

`[-90:90]` (default) | 1-by-*Q* real-valued row vector | *Q*-by-1 real-valued column vector

Specify the elevation angles used to define the angular coordinates of the **RCS pattern (m^2)** parameter. Specify elevation angles as a length *Q* vector. Units are degrees. *Q* must be greater than two. This parameter determines the incident elevation arrival angle of any element of the cross-section patterns.

RCS pattern (m^2) — Radar cross-section pattern

`ones(181,361)` (default) | *Q*-by-*P* real-valued matrix | *Q*-by-*P*-by-*M* real-valued array | 1-by-*P* real-valued vector | *M*-by-*P* real-valued matrix

Radar cross-section pattern, specified as a *Q*-by-*P* real-valued matrix or a *Q*-by-*P*-by-*M* real-valued array.

- *Q* is the length of the vector in the **Elevation angles (deg)** parameter.
- *P* is the length of the vector in the **Azimuth angles (deg)** parameter.

- *M* is the number of target patterns. The number of patterns corresponds to the number of signals passed into the input port X. You can, however, use a single pattern to model multiple signals reflecting from a single target.

You can, however, use a single pattern to model multiple signals reflecting from a single target. Pattern units are square-meters.

Pattern units are square-meters.

Fluctuation model — Target fluctuation model

`Nonfluctuating` (default) | `Swerling1` | `Swerling2` | `Swerling3` | `Swerling4`

Specify the statistical model of the target as either `Nonfluctuating`, `Swerling1`, `Swerling2`, `Swerling3`, or `Swerling4`. When you set this parameter to a value other than `Nonfluctuating`, you then set radar cross-sections parameters using the `Update` input port.

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — Operating frequency

`3e8` (default) | positive scalar

Carrier frequency of the signal that reflects from the target, specified as a positive scalar. Units are in hertz.

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><td rowspan="2">Block Simulation</td><td colspan="3">Simulation Behavior</td></tr>
<tr><td><code>Normal</code></td><td><code>Accelerator</code></td><td><code>Rapid Accelerator</code></td></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

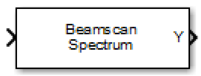
Introduced in R2016a

See Also

`phased.BackscatterRadarTarget`

Beamscan Spectrum

Beamscan spatial spectrum estimator



Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The Beamscan Spectrum block estimates the 2-D spatial spectrum of incoming narrowband signals by scanning a range of azimuth and elevation angles using a narrowband conventional beamformer. The block optionally calculates the direction of arrival of a specified number of signals by locating peaks of the spectrum.

Ports

Input

X — Received signal
M-by-*N* complex-valued matrix

Received signal, specified as an *M*-by-*N* complex-valued matrix. The quantity *M* is the length of the signal, the number of sample values contained in the signal. The quantity *N* is the number of sensor elements in the array.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Output

Y — Beamscan 2-D spatial spectrum
non-negative real-valued *P*-by-*Q* matrix

2Magnitude of the estimated 2-D spatial spectrum, returned as a non-negative, returned as a real-valued *P*-by-*Q* matrix. Each entry represents the magnitude of the estimated MUSIC spatial spectrum. Each entry corresponds to an angle specified by the **Azimuth scan angles (deg)** and **Elevation scan angles (deg)** parameters. *P* equals the length of the vector specified in **Azimuth scan angles (deg)** and *Q* equals the length of the vector specified in **Elevation scan angles (deg)**.

Data Types: `double`

Ang — Directions of arrival
non-negative, real-valued 2-by-*L* matrix

Directions of arrival of the signals, returned as a real-valued 2-by-*L* matrix. *L* is the number of signals specified by the **Number of signals** parameter. The direction of arrival angle is defined by the azimuth and elevation angles of the source with respect to the array local coordinate system. The first row of the matrix contains the azimuth angles and the second row contains the elevation angles. If the object cannot identify peaks in the spectrum, it will return `NaN`. Angle units are in degrees.

Dependencies

To enable this output port, select the **Enable DOA output** check box.

Data Types: `double`

Parameters

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Number of bits in phase shifters — Number of phase shift quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Forward-backward averaging — Enable forward-backward averaging
off (default) | on

Select this parameter to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold structure.

Azimuth scan angles (deg) — Scan angles in azimuth direction
`-180:180` (default) | real-valued vector

Scan angles in azimuthal direction, specified as a real-valued vector. The angles must lie be between –180° and 180°, inclusive. You must specify the angles in ascending order. Units are in degrees.

Data Types: `double`

Elevation scan angles (deg) — Scan angles in elevation direction
`-90:90` (default) | real-valued vector

Scan angles in elevation direction, specified as a real-valued vector. The angles must lie be between –90° and 90°, inclusive. You must specify the angles in ascending order. Units are in degrees.

Data Types: `double`

Enable DOA output — Output directions of arrival through output port
off (default) | on

Select this parameter to output the signals directions of arrival (DOA) through the **Ang** output port.

Number of signals — Expected number of arriving signals
`1` (default) | positive integer

Specify the expected number of signals for DOA estimation as a positive scalar integer.

Dependencies

To enable this parameter, select the **Enable DOA output** check box.

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-Q vector. Q must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-P row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-P row vector. P must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.

- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles

specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

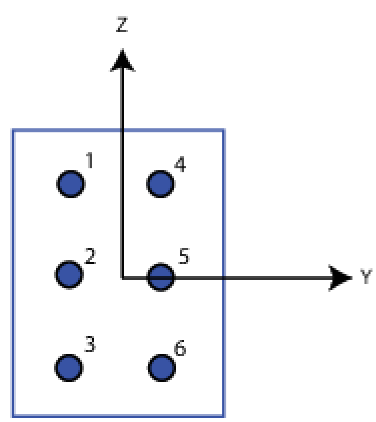
Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of elements in each row and column.

For a URA, array elements are indexed from top to bottom along the leftmost array column, and continued to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
`x`	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
`y`	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
`z`	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Data Types: `double`

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. If the parameter value is a matrix, each

column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

To enable this parameter, set **Geometry** to `Conformal Array`.

Data Types: `double`

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Version History

Introduced in R2014b

See Also

`phased.BeamscanEstimator2D` | `phased.ConformalArray` | `phased.UCA` | `phased.ULA` | `phased.URA`

Beamspace ESPRIT DOA

Beamspace ESPRIT direction of arrival (DOA) estimator for ULA

Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The Beamspace ESPRIT DOA block estimates the direction of arrival of a specified number of narrowband signals incident on a uniform linear array using the estimation of signal parameters via rotational invariance technique (ESPRIT) algorithm in beamspace.

Ports

Input

Port_1 — Input signal
N-by-*M* complex-valued matrix

Input signal, specified as an *N*-by-*M* complex-valued matrix. *N* corresponds to the number of samples. *M* corresponds to the number data channels.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`

Output

Ang — Estimated broadside direction-of-arrival angles
M-by-1 real-valued vector

Estimated broadside direction-of-arrival angles, returned as an *M*-by-1 real-valued vector. Units are in degrees.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Number of signals — Number of signals

`1` (default) | positive integer scalar

Specify the number of signals as a positive integer scalar.

Spatial smoothing — Enable spatial smoothing
`0` (default) | non-negative integer

Specify the amount of averaging used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each increase in smoothing handles one extra coherent source, but reduces the effective number of elements by one. The maximum value of this parameter is $N - 2$, where N is the number of sensors in the ULA.

Type of least squares method — Type of least squares method

`TLS` (default) | `LS`

Specify the least squares method used for ESPRIT as one of `TLS` or `LS` where `TLS` refers to total least squares and `LS`refers to least squares.

Beam fan center direction (deg) — Beam fan center direction

`0` (default)

Specify the direction of the center of the beam fan, in degrees, as a real scalar value between –90° and 90°.

Source of number of beams — Source of number of beams

`Auto` (default) | `Property`

Specify the source of the number of beams as one of `Auto` or `Property`. If you set this parameter to `Auto`, the number of beams equals $N - L$, where N is the number of array elements and L is the value of **Spatial smoothing**.

Number of beams — Number of beams

2

Specify the number of beams as a positive scalar integer. The lower the number of beams, the greater the reduction in computational cost.

Dependencies

This parameter appears when you set **Source of number of beams** to `Property`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><th rowspan="2">Block Simulation</th><th colspan="3">Simulation Behavior</th></tr>
<tr><th><code>Normal</code></th><th><code>Accelerator</code></th><th><code>Rapid Accelerator</code></th></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Element Parameters

Element type — Array element types
Isotropic Antenna (default) | Cardioid Antenna | Cosine Antenna | Custom Antenna | Gaussian Antenna | Sinc Antenna | Omni Microphone | Custom Microphone

Antenna or microphone type, specified as one of the following:

- Isotropic Antenna
- Cardioid Antenna
- Cosine Antenna
- Custom Antenna
- Gaussian Antenna
- Sinc Antenna
- Omni Microphone
- Custom Microphone

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
[0,1e20] (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form [LowerBound,UpperBound]. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to Isotropic Antenna, Cosine Antenna, or Omni Microphone.

Baffle the back of the element — Set back response of an Isotropic Antenna element or an Omni Microphone element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to Isotropic Antenna or Omni Microphone.

Null axis direction — Null axis direction
-x (default) | +x | +y | -y | +z | -z

Dependencies

To enable this parameter, set **Element type** to Cardioid Antenna.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Align element normal with array normal — Align element normal with array normal
on (default) | off

Dependencies

This parameter is enabled when **Element type** is set to `Custom Antenna`.

Radiation pattern beamwidth (deg) — Radiation pattern beamwidth
`[10,10]` (default)

Dependencies

This parameter is enabled when **Element type** is set to `Gaussian Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

ULA Parameters

Number of elements — Number of array elements in U
2 (default) | positive integer greater than or equal to two

The number of array elements for ULA arrays, specified as an integer greater than or equal to two.

Example: `11`

Data Types: `double`

Element spacing — Distance between ULA elements
`0.5` (default) | positive scalar

Distance between adjacent ULA elements, specified as a positive scalar. Units are in meters.

Example: `1.5`

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Taper — ULA array taper
`1` (default) | complex-valued vector

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Example: [0.5;1;0.5]

Data Types: `double`

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create a ULA array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.ULA('NumElements',13)`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Version History

Introduced in R2014b

See Also

`phased.BeamspaceESPRITEstimator`

CFAR Detector

Constant false alarm rate (CFAR) detector

Libraries:
Phased Array System Toolbox / Detection

Description

The CFAR Detector block implements a one-dimensional constant false-alarm rate (CFAR) detector. Detection processing is performed on selected elements (called cells) of the input data. A detection is declared when an image cell value exceeds a threshold. To maintain a constant false alarm-rate, the threshold is set to a multiple of the image noise power. The detector estimates noise power for a cell-under-test (CUT) from surrounding cells using one of three cell averaging methods, or an order statistics method. The cell-averaging methods are cell averaging (CA), greatest-of cell averaging (GOCA), or smallest-of cell averaging (SOCA).

For more information about CFAR detectors, see [1].

For each test cell, the detector:

1 estimates the noise statistic from the cell values in the training band surrounding the CUT cell.
2 computes the threshold by multiplying the noise estimate by the threshold factor.
3 compares the CUT cell value to the threshold to determine whether a target is present or absent. If the value is greater than the threshold, a target is present.

Ports

Input

X — Cell matrix
real-valued *M*-by-1 column vector | real-valued *M*-by-*N* matrix

Cell matrix, specified as a real-valued *M*-by-1 column vector or a real-valued *M*-by-*N* matrix.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`

Idx — Index of cells under test
real-valued length-*D* vector

Cells-under-test (CUT), specified as a real-valued length-*D* vector. Indices specify the input elements or cells under test on which to perform detection processing. When `X` is a vector, `Idx` specifies the element. When `X` is a matrix, `Idx` specifies the row of the element. The same index applies to all columns of the matrix. Detection is performed independently along each column of Xww for the indices specified in `Idx`.

Data Types: `double`

K — Threshold factor
positive scalar

Threshold factor, used to calculate the detection threshold, specified as a positive scalar.

Dependencies

To enable this port, set the **Threshold factor method** parameter to `Input port`.

Data Types: `double`

Output

Y — Detection results
D-by-1 vector | *D*-by-*N* matrix | *1*-by-*L* | 2-by-*L* matrix

The format of `Y` depends on the **Output format** property.

- When **Output format** is `'Cut result'`, `Y` is a *D*-by-1 vector or a *D*-by-*N* matrix containing logical detection results. *D* is the length of `Idx` and *N* is the number of columns of `X`. The rows of `Y` correspond to the rows in `Idx`. For each row, `Y` contains `1` in a column if there is a detection in the corresponding column of `X`. Otherwise, `Y` contains a `0`.
- When **Output format** is `'Detection report'`, `Y` is a 1-by-*L* vector or a 2-by-*L* matrix containing detections indices. *L* is the number of detections found in the input data. When `X` is a column vector, `Y` contains the index for each detection in `X`. When `X` is a matrix, `Y` contains the row and column indices of each detection in `X`. Each column of `Y` has the form `[detrow;detcol]`. When the **Source of number of detections** property is set to `'Property'`, *L* equals the value of the **Maximum number of detections** parameter. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

Data Types: `double`

Th — Detection threshold
scalar

Detection threshold applied to cells under test, returned as a scalar.

- When **Output format** is `'CUT result'`, `Th` port outputs the detection threshold whenever an element of `Y` is `1` and `NaN` whenever an element of `Y` is `0`. The output of `Th` has the same size as `Y`.
- When **Output format** is `'Detection index'`, the `Th` port outputs a detection threshold for each corresponding detection in `Y`.When the **Source of the number of detections** parameter is set to `'Property'`, the number of detections is set by the **Maximum number of detections** parameter. If the number of actual detections is less than this value, the columns without detections are set to `NaN`.

Dependencies

To enable this port, select the **Output detection threshold** check box.

Data Types: `double`

N — Estimated noise power
positive scalar

Estimated noise power for each detected cell under test, returned as a positive scalar.

- When **Output format** is `'CUT result'`, `N` returns a noise power estimate when `Y` is `1` and `NaN` whenever `Y` is zero. The output from port `N` has the same size as `Y`.
- When **Output format** is `'Detection index'`, `M` returns a noise power estimate for each corresponding detection in `Y`. When the **Source of the number of detections** property is set to `'Property'`, *L* equals the value of the **Maximum number of detections** parameter. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

Dependencies

To enable this port, select the **Output estimated noise power** check box.

Data Types: `double`

Parameters

CFAR algorithm — CFAR algorithm

CA (default) | `GOCA` | `SOCA` | `OS`

Specify the CFAR detection algorithm using one of the values

`CA`	Cell-averaging
`GOCA`	Greatest-of cell averaging
`OS`	Order statistic
`SOCA`	Smallest-of cell averaging

Number of guard cells — Number of guard cells

2 (default)

Specify the number of guard cells used in training as an even integer. This parameter specifies the total number of cells on both sides of the cell under test.

Number of training cells — Number of training cells

2 (default)

Specify the number of training cells used in training as an even integer. Whenever possible, the training cells are equally divided before and after the cell under test.

Rank of order statistic — Rank of order statistic

1 (default)

Specify the rank of the order statistic as a positive integer scalar. The value must be less than or equal to the value of **Number of training cells**.

Dependencies

This parameter appears when **CFAR algorithm** is set to `OS`.

Threshold factor method — Methods of obtaining threshold factor

`Auto` (default) | `Input port` | `Custom`

Specify whether the threshold factor comes from an automatic calculation, the **Custom threshold factor** parameter, or an input argument. Values of this parameter are:

`Auto`	The application calculates the threshold factor automatically based on the desired probability of false alarm specified in the **Probability of false alarm** parameter. The calculation assumes each independent signal in the input is a single pulse coming out of a square law detector with no pulse integration. The calculation also assumes the noise is white Gaussian.
`Custom`	The **Custom threshold factor** parameter specifies the threshold factor.
`Input port`	Threshold factor is set using the input port `K` . This port appears only when **Threshold factor method** is set to `Input port` .

Probability of false alarm — Desired probability of false alarm

`0.1` (default)

Specify the desired probability of false alarm as a scalar between 0 and 1 (not inclusive).

Dependencies

This parameter appears only when you set **Threshold factor method** to `Auto`.

Custom threshold factor — Custom threshold factor

`1` (default)

Specify the custom threshold factor as a positive scalar.

Dependencies

This parameter appears only when you set **Threshold factor method** to `Custom`.

Output format — Format of detection results

`CUT result` (default) | `Detection index`

Specify the format of detection results returned in output port `Y` as `CUT result` or `Detection index`.

- When set to `CUT result`, the results are logical detection values (`1` or `0`) for each tested cell. `1` indicates that the value of the tested cell exceeds a detection threshold.
- When set to `Detection index`, the results form a vector or matrix containing the indices of tested cells which exceed a detection threshold.

Output detection threshold — Output detection threshold

off (default) | on

Select this check box to create an output port `Th` containing the detection threshold.

Output estimated noise power — Output estimated noise power

off (default) | on

Select this check box to create an output port `N` containing the estimated noise.

Source of the number of detections — Source of the number of detections

`Auto` (default) | `Property`

Specify the source of the number of detections as `Auto` or `Property`. When you select `Auto`, the number of detection indices reported is the total number of cells under test that have detections. If you select `Property`, the number of reported detections is determined by the value of the **Maximum number of detections** parameter.

Dependencies

To enable this parameter, set the **Output format** parameter to `Detection index`.

Maximum number of detections — Maximum number of detections to report

`1` (default)

Specify the maximum number of detection indices to report as a positive integer.

Dependencies

To enable this parameter, set the **Output format** parameter to `Detection index` and the **Source of the number of detections** parameter to `Property`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also

Functions
`npwgnthresh` | `rocpfa`

Objects
`phased.CFARDetector` | `phased.CFARDetector2D`

Blocks
2-D CFAR Detector

2-D CFAR Detector

Two-dimensional constant false alarm rate (CFAR) detector

Libraries:
Phased Array System Toolbox / Detection

Description

The 2-D CFAR Detector block implements a constant false-alarm rate detector for two dimensional image data. A detection is declared when an image cell value exceeds a threshold. To maintain a constant false alarm-rate, the threshold is set to a multiple of the image noise power. The detector estimates noise power from neighboring cells surrounding the cell-under-test (CUT) using one of three cell averaging methods, or an order statistics method. The cell-averaging methods are cell-averaging (CA), greatest-of cell averaging (GOCA), or smallest-of cell averaging (SOCA).

For each test cell, the detector:

1. estimates the noise statistic from the cell values in the training band surrounding the CUT cell.
2. computes the threshold by multiplying the noise estimate by the threshold factor.
3. compares the CUT cell value to the threshold to determine whether a target is present or absent. If the value is greater than the threshold, a target is present.

Ports

Input

X — Input image
real *M*-by-*N* matrix | real *M*-by-*N*-by-*P* array

Input image, specified as a real *M*-by-*N* matrix or real *M*-by-*N*-by-*P* array. *M* and *N* represent the rows and columns of the matrix. Each page is a different 2-D signal.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

idx — Location of test cells
2-by-*L* matrix of positive integers

Location of test cells, specified as a *2*-by-*L* matrix of positive integers, where *L* is the number of test cells. Each column of `idx` specifies the row and column index of a CUT cell. The locations of CUT cells are restricted so that their training regions lie completely within the input images.

Data Types: `double`

K — Detection threshold factor
positive scalar

Threshold factor used to calculate the detection threshold, specified as a positive scalar.

Dependencies

To enable this port, set the **Threshold factor method** parameter to `'Input port'`

Data Types: `double`

Output

Y — Detection results
logical matrix (default) | real-valued matrix

Detection results, whose format depends on the `Output Format` property

- When `OutputFormat` is `'Cut result'`, `Y` is a *D*-by-*P* matrix containing logical detection results for cells under test. *D* is the length of `cutidx` and *P* is the number of pages of `X`. The rows of `Y` correspond to the rows of `cutidx`. For each row, `Y` contains `1` in a column if there is a detection in the corresponding cell in `X`. Otherwise, `Y` contains a `0`.
- When `OutputFormat` is `Detection index`, `Y` is a *K*-by-*L* matrix containing detections indices. *K* is the number of dimensions of `X`. *L* is the number of detections found in the input data. When `X` is a matrix, `Y` contains the row and column indices of each detection in `X` in the form `[detrow;detcol]`. When `X` is an array, `Y` contains the row, column, and page indices of each detection in `X` in the form `[detrow;detcol;detpage]`. When the `NumDetectionsSource` property is set to `'Property'`, *L* equals the value of the `NumDetections` property. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

Data Types: `double`

Th — Computed detection threshold
real-valued matrix

Computed detection threshold for each detected cell, returned as a real-valued matrix. `Th` has the same dimensions as `Y`.

- When `OutputFormat` is `'CUT result'`, `Th` returns the detection threshold whenever an element of `Y` is `1` and `NaN` whenever an element of `Y` is `0`.
- When `OutputFormat` is `Detection index`, `th` returns a detection threshold for each corresponding detection in `Y`. When the `NumDetectionsSource` property is set to `'Property'`, *L* equals the value of the `NumDetections` property. If the number of actual detections is less than this value, columns without detections are set to `NaN`.

Dependencies

To enable this port, select the **Output detection threshold** checkbox.

Data Types: `double`

N — Estimated noise power
real-valued matrix

Estimated noise power for each detected cell, returned as a real-valued matrix. noise has the same dimensions as Y.

- When OutputFormat is 'CUT result', noise returns the noise power whenever an element of Y is 1 and NaN whenever an element of Y is 0.
- When OutputFormat is 'Detection index', noise returns a noise power for each corresponding detection in Y. When the NumDetectionsSource property is set to 'Property', *L* equals the value of the NumDetections property. If the number of actual detections is less than this value, columns without detections are set to NaN.

Dependencies

To enable this port, select the **Output estimated noise power** checkbox.

Data Types: double

Parameters

CFAR algorithm — Noise power estimation algorithm
CA (default) | GOCA | SOCA | OS

Noise power estimation algorithm, specified as CA, GOCA, SOCA, or OS. For CA, GOCA, SOCA, the noise power is the sample mean derived from the training band. For OS, the noise power is the *k*th cell value obtained from a numerical ordering of all training cell values. Set *k* by the **Rank of order statistic** parameter. See “Training cells” on page 3-68.

Averaging Method	Description
CA — Cell-averaging algorithm	Computes the sample mean of all training cells surrounding the CUT cell.
GOCA — Greatest-of cell-averaging algorithm	Splits the 2-D training window surrounding the CUT cell into left and right halves. Then, the algorithm computes the sample mean for each half and selects the largest mean.
SOCA — Smallest-of cell-averaging algorithm	Splits the 2-D training window surrounding the CUT cell into left and right halves. Then, the algorithm computes the sample mean for each half and selects the smallest mean.
OS — Order statistic algorithm	Sorts training cells in ascending order of numeric values. Then the algorithm selects the *k*th value from the list. *k* is the rank specified by the Rank parameter.

Rank of order statistic — Rank of order statistic
1 (default) | positive integer

Specify the rank of the order statistic used in the 2-D CFAR algorithm as a positive integer. The value of this parameter must lie between 1 and N_{train}, where N_{train} is the number of training cells. A value of 1 selects the smallest value in the training region.

Dependencies

To enable this parameter, set the **CFAR Algorithm** parameter to `OS`.

Size in cells of the guard region band — Widths of guard band
`[1,1]` (default) | nonnegative integer scalar | 2-element vector of positive integers

The number of row and column guard cells on each side of the cell under test as nonnegative integers. The first element specifies the guard band size along the row dimension. The second element specifies the guard band size along the row dimension. Specifying **Size in cells of the guard region band** as a scalar is equivalent to specifying a vector with the same value for both dimensions. For example, a value of `[1 1]`, indicates that there is a one-guard-cell-wide region surrounding each CUT cell.

Size in cells of the training region band — Widths of training band
`[1,1]` (default) | nonnegative integer scalar | 2-element vector of positive integers

Size in cells of the training region band, specified as a nonnegative integer or 1-by-2 matrix of nonnegative integers. The first element specifies the training band size along the row dimension, and the second along the column dimension. Specifying **Size in cells of the training region band** as a scalar is equivalent to specifying a vector with the same value for both dimensions. For example, a value of `[1 1]` indicates that there is a one-training-cell-wide region surrounding the guard region for each cell under test.

Threshold factor method — Method to determine threshold factor
`Auto` (default) | `Input port` | `Custom`

Method to determine threshold factor, specified as `Auto`, `Input port`, or `Custom`.

- When you choose `Auto`, the threshold factor is determined from the estimated noise statistic and the probability of false alarm.
- When you choose `Input Port`, set the threshold factor using the `K` input port.
- When you choose `Custom`, set the threshold factor using the **Custom threshold factor** parameter.

Custom threshold factor — Custom threshold factor
`1` (default) | positive scalar

Custom threshold factor, specified as a positive scalar.

Dependencies

To enable this parameter, set the **Threshold factor method** parameter to `Custom`.

Probability of false alarm — Probability of false alarm
`0.1` (default) | real scalar between 0 and 1

Probability of false alarm, specified as a real scalar between 0 and 1. You can calculate the threshold factor from the required probability of false alarm.

Dependencies

To enable this parameter, set the **Threshold factor method** property to `Auto`.

OutputFormat — Format of detection results
`CUT result` (default) | `Detection index`

Format of detection results, specified as `CUT result` or `Detection index`.

- When set to `'CUT result'`, the detection results are logical detection values (`1` or `0`) for each tested cell.
- When set to `'Detection index'`, the results form a vector or matrix containing the indices of tested cells that exceed a detection threshold.

Output threshold detection — Enable detection threshold output
off (default) | on

Select this check box to enable the output of detection thresholds via the `Th` output port.

Output estimated noise power — Enable detection threshold output
off (default) | on

Select this check box to enable the output of estimated noise power via the `N` output port.

Source of the number of detections — Source of the number of detections to report
`Auto` (default) | `Property`

Source of the number of detections, specified as `Auto` or `Property`. When you select `Auto`, the number of detection indices reported is the total number of cells under test that have detections. If you select `Property`, the number of reported detections is determined by the value of the **Maximum number of detections** parameter.

Dependencies

To enable this parameter, set the **Output format** parameter to `Detection index`.

Data Types: `char`

Maximum number of detections — Maximum number of detection indices to report
`1` (default) | positive integer

Maximum number of detection indices to report, specified as a positive integer.

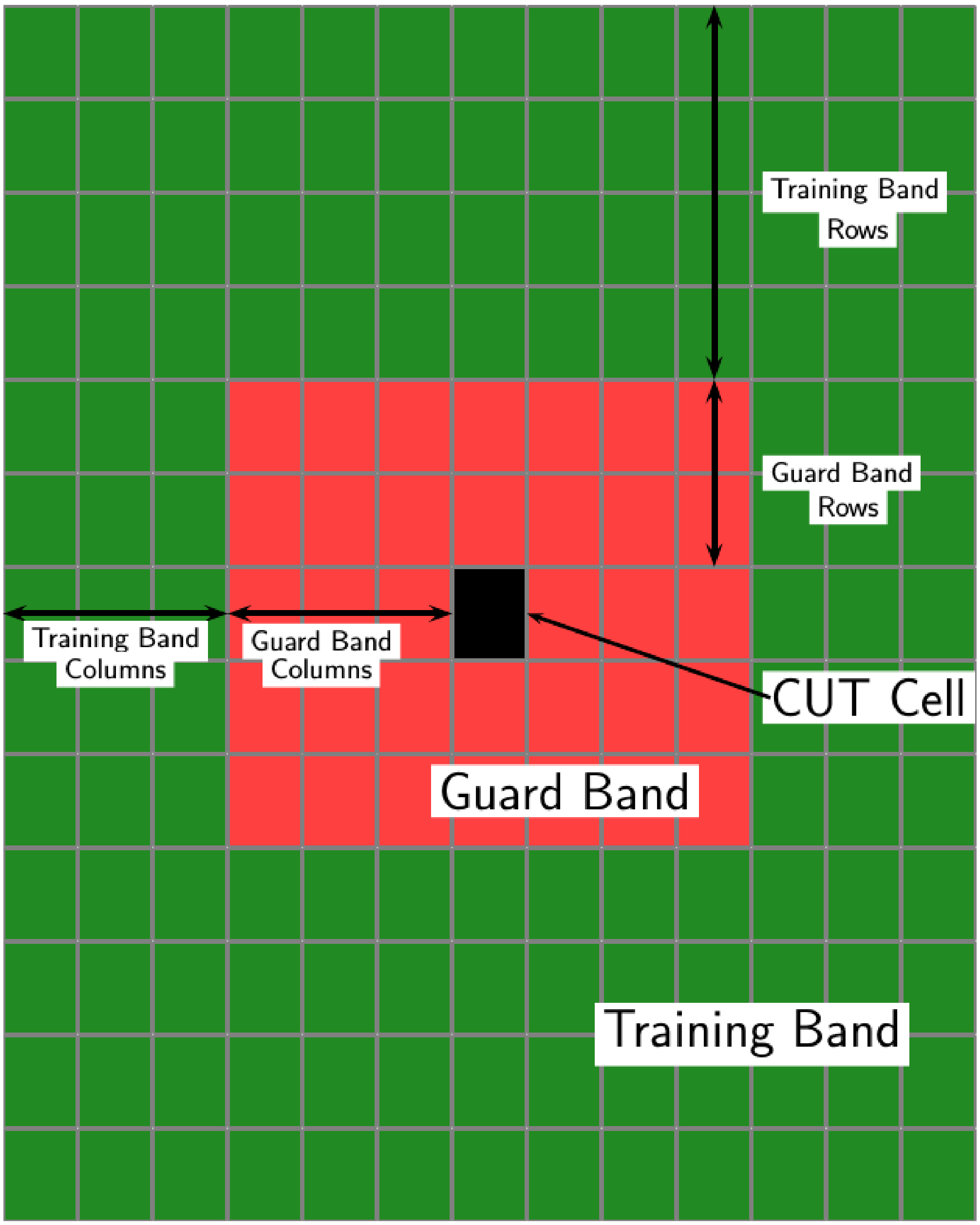
Dependencies

To enable this parameter, set the **Output format** parameter to `Detection index` and the **Source of the number of detections** parameter to `Property`.

Data Types: `double`

Algorithms

CFAR 2-D requires an estimate of the noise power. Noise power is computed from cells that are assumed not to contain any target signal. These cells are the training cells. Training cells form a band around the cell-under-test (CUT) cell but may be separated from the CUT cell by a guard band. The detection threshold is computed by multiplying the noise power by the threshold factor.



For GOCA and SOCA averaging, the noise power is derived from the mean value of one of the left or right halves of the training cell region.

Because the number of columns in the training region is odd, the cells in the middle column are assigned equally to either the left or right half.

When using the order-statistic method, the rank cannot be larger than the number of cells in the training cell region, N_{train}. You can compute N_{train}.

- N_{TC} is the number of training band columns.
- N_{TR} is the number of training band rows.
- N_{GC} is the number of guard band columns.
- N_{GR} is the number of guard band rows.

The total number of cells in the combined training region, guard region, and CUT cell is $N_{total} = (2N_{TC} + 2N_{GC} + 1)(2N_{TR}+ 2N_{GR} + 1)$.

The total number of cells in the combined guard region and CUT cell is $N_{guard} = (2N_{GC} + 1)(2N_{GR} + 1)$.

The number of training cells is $N_{train} = N_{total} - N_{guard}$.

By construction, the number of training cells is always even. Therefore, to implement a median filter, you can choose a rank of $N_{train}/2$ or $N_{train}/2 + 1$.

Version History

Introduced in R2016b

See Also

Functions
`npwgnthresh` | `rocpfa`

Objects
`phased.CFARDetector` | `phased.CFARDetector2D`

Blocks
CFAR Detector

Custom FM Waveform

Custom FM pulse waveform

Libraries:
Phased Array System Toolbox / Waveforms

Description

The Custom FM Waveform block generates a custom FM pulse waveform with specified pulse width, pulse repetition frequency (PRF), and sweep bandwidth. The block outputs an integer number of pulses or an integral number of samples.

Ports

Input

PRFIdx — PRF Index
positive integer

Index to select the pulse repetition frequency (PRF), specified as a positive integer. The index selects the PRF from the predefined vector of values specified by the **Pulse repetition frequency (Hz)** parameter.

Example: 4

Dependencies

To enable this port, select **Enable PRF selection input**.

Data Types: `double`

FreqOffset — Frequency offset
scalar

Frequency offset in Hz, specified as a scalar.

Example: `2e3`

Dependencies

To enable this port, set **Source of Frequency Offset** to `Input port`.

Data Types: `double`

Output

Y — Pulse waveform
complex-valued vector

Pulse waveform samples, returned as a complex-valued vector.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, returned as a positive scalar.

Dependencies

To enable this port, set the **Output signal format** parameter to `Pulses` and then select the **Enable PRF output** parameter.

Data Types: `double`

Coeff — Matched filter coefficients
vector | matrix

Matched filter coefficients, returned as a vector or matrix.

Dependencies

To enable this port, select **Enable Matched Filter Coefficients Output**.

Data Types: `double`

Parameters

To edit block parameters interactively, use the **Property Inspector**. From the Simulink Toolstrip, on the **Simulation** tab, in the **Prepare** gallery, select **Property Inspector**.

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.

Programmatic Use
Block Parameter:`SampleRate`
Type:double
Values:positive scalar
Default:`1e6`

Method to specify pulse duration — Pulse duration as time or duty cycle
`Pulse width` (default) | `Duty cycle`

Method to set the pulse duration, specified as `Pulse width` or `Duty cycle`. When you set this parameter to `Pulse width`, the pulse duration is set using the **Pulse width (s)** parameter. When you set this parameter to `Duty cycle`, the pulse duration is computed from the values of the **Pulse repetition frequency (Hz)** and **Duty Cycle** parameters.

Programmatic Use
Block Parameter:`DurationSpecification`
Type:string
Values:string
Default:`'Pulse width'`

Pulse width (s) — Time duration of pulse
`50e-6` (default) | positive scalar

The duration of each pulse, specified as a positive scalar. Set the product of **Pulse width (s)** and **Pulse repetition frequency** to be less than or equal to one. This restriction ensures that the pulse width is smaller than the pulse repetition interval. Units are in seconds.

Example: `300e-6`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Pulse width`.

Programmatic Use
Block Parameter:`PulseWidth`
Type:double
Values:string
Default:`50e-6`

Duty cycle — Waveform duty cycle
`0.5` (default) | scalar in the range *[0,1]*

Waveform duty cycle, specified as a scalar in the range *[0,1]*.

Example: `0.7`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Duty cycle`.

Programmatic Use
Block Parameter:`DutyCycle`
Type:double
Values:positive scalar
Default:`1e6`

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1e4` (default) | positive scalar

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The value of **Pulse repetition frequency (Hz)** must satisfy these constraints:

- The product of **Pulse width** and **Pulse repetition frequency (Hz)** must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of **Pulse repetition frequency** must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* by using block parameter settings alone or in conjunction with the input port, `PRFIdx`.

- When the **Enable PRF selection input** parameter is not selected, set the *PRF* using block parameters.
  - To implement a constant *PRF*, specify **Pulse repetition frequency (Hz)** as a positive scalar.
  - To implement a staggered *PRF*, specify **Pulse repetition frequency (Hz)** as a row vector with positive values. After the waveform reaches the last element of the vector, the process continues cyclically with the first element of the vector. When *PRF* is staggered, the time between successive output pulses cycles through the successive values of the *PRF* vector.
- When the **Enable PRF selection input** parameter is selected, you can implement a selectable *PRF* by specifying **Pulse repetition frequency (Hz)** as a row vector with positive real-valued entries. But this time, when you execute the block, select a *PRF* by passing an index into the *PRF* vector into the `PRFIdx` port.

In all cases, the number of output samples is fixed when you set the **Output signal format** to `Samples`. When you use a varying *PRF* and set **Output signal format** to `Pulses`, the number of output samples can vary.

Programmatic Use

Block Parameter:PRF
Type:double
Values:positive scalar
Default:`1e6`

Enable PRF selection input — Select predefined PRF

off (default) | on

Select this parameter to enable the `PRFIdx` port.

- When enabled, pass in an index into a vector of predefined PRFs. Set predefined PRFs using the **Pulse repetition frequency (Hz)** parameter.
- When not enabled, the block cycles through the vector of PRFs specified by the **Pulse repetition frequency (Hz)** parameter. If **Pulse repetition frequency (Hz)** is a scalar, the PRF is constant.

Programmatic Use

Block Parameter:`PRFSelectionInputPort`
Type:logical
Values:positive scalar
Default:`off`

Source of simulation sample time — Source of simulation sample time

`Derive from waveform parameters` (default) | `Inherit from Simulink engine`

Source of simulation sample time, specified as `Derive from waveform parameters` or `Inherit from Simulink engine`. When set to `Derive from waveform parameters`, the block runs at a variable rate determined by the PRF of the selected waveform. The elapsed time is variable. When set to `Inherit from Simulink engine`, the block runs at a fixed rate so the elapsed time is a constant.

Dependencies

To enable this parameter, select the **Enable PRF selection input** parameter.

Programmatic Use
Block Parameter:`SimulationTimeSource`
Type:enum
Values:`Derive from waveform parameters`, `Inherit from Simulink engine`
Default:`Derive from waveform parameters`

Frequency modulation — Wavefform requency modulation function
length-M real-valued vector | function handle | cell array

Waveform frequency modulation function, specified as a length-M real-valued vector, function handle, or cell array.

- If the `FrequencyModulation` property is a vector, it specifies samples of the instantaneous frequency at M points as [f_1, f_2, ..., f_M]. The waveform sweeps the specified frequencies such for the k^{th} pulse with start time t_k and duration T_k, the instantaneous frequency at time

  $$t_m = t_k + (m - 1)T_k/(M - 1)$$

  is equal to f_m where $t_k \leqq t_m \leqq t_k + T_k$ and $m = 1...M$. The instantaneous frequencies between time t_m and t_{m+1} are found by linearly interpolating between f_m and f_{m+1}. The resulting custom FM waveform can be considered as a piecewise LFM waveform consisting of M-1 LFM sections of equal duration.
- If the `FrequencyModulation` property is a function handle, the function must have the following syntax: `f = fmFcn(t)` where f is the instantaneous frequency at time t. t is the time at which to compute the instantaneous frequency. The values in t are between 0 and the pulse width.
- If the `FrequencyModulation` property is a cell array, then the first cell must be a function handle as specified above. The remaining entries in the cell array are the additional input arguments to the function, if any.

Programmatic Use
Block Parameter:`FrequencyModulation`
Type:char, string, function
Values:char. string
Default:`Hyperbolic`

Data Types: `char` | `string`

Envelope function — FM signal amplitude envelope
`Rectangular` (default) | `Gaussian` | `Hamming` | `Chebyshev` | `Hann` | `Kaiser` | `Taylor`

FM signal amplitude envelope function, specified as `Rectangular`, `Gaussian`, `Hamming`, `Chebyshev`, `Hann`, `Kaiser`, or `Taylor`.

Programmatic Use
Block Parameter:`Envelope`
Type:enum
Values:`Rectangular`, `Gaussian`, `Hamming`, `Chebyshev`, `Hann`, `Kaiser`, or `Taylor`
Default:`Rectangular`

Source of Frequency Offset — Source of frequency offset
`Property` (default) | `Input port`

Source of frequency offset, specified as `Property` or `Input port`.

- When set to `Property`, the offset is determined by the value of the **Frequency Offset** parameter.
- When set to `Input port`, the offset is determined by the value of the **FreqOffset** port.

Programmatic Use
Block Parameter:`FrequencyOffsetSource`
Type:enum
Values:`Property`, `Input Port`
Default:`Property`

Frequency Offset (Hz) — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `2e3`

Dependencies

To enable this parameter set the **Source of Frequency Offset** parameter to `Input port`.

Programmatic Use
Block Parameter:`FrequencyOffset`
Type:double
Values:`scalar`
Default:`0`

Output signal format — Format of the output signal
`Pulses` (default) | `Samples`

The format of the output signal, specified as `Pulses` or `Samples`.

If you set this parameter to `Samples`, the output of the block consists of multiple samples. The number of samples is the value of the **Number of samples in output** parameter.

If you set this parameter to `Pulses`, the output of the block consists of multiple pulses. The number of pulses is the value of the **Number of pulses in output** parameter.

Programmatic Use
Block Parameter:`OutputFormat`
Type:enum
Values:`Pulses` `Samples`
Default:`Pulses`

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use
Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Number of pulses in output — Number of pulses in output
`1` (default) | positive integer

Number of pulses in the block output, specified as a positive integer.

Example: 2

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Pulses`.

Programmatic Use
Block Parameter:`NumPulses`
Type:double
Values:`positive scalar`
Default:`1`

Data Types: `double`

Enable PRF Output — Enable output of PRF
`off` (default) | `on`

Select this parameter to enable the PRF output port.

Dependencies

To enable this parameter, set **Output signal format** to `Pulses`.

Programmatic Use
Block Parameter:`PRFOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Enable Matched Filter Coeficients Output — Enable output of matched filter coefficients
`off` (default) | `on`

Select this parameter to enable the `Coeff` output port.

Programmatic Use
Block Parameter:`CoefficientOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use

Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

Introduced in R2023a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

`phased.CustomFMWaveform` | `phased.NonlinearFMWaveform`

Data Cube Slicer

Slice data cube along specified dimensions

Libraries:
Phased Array System Toolbox / Space-Time Adaptive Processing

Description

The Data Cube Slicer block slices a data cube through a specified plane. The input is a data cube of dimensions M-by-Q-by-N. The first dimension is range, or fast time. The second dimension is angle, or channels. The third dimension is Doppler, or slow time.

Ports

Input

X — Input data cube
M-by-Q-by-N complex-valued array

Input data cube, specified as an M-by-Q-by-N complex-valued matrix.

Data Types: `double`
Complex Number Support: Yes

Idx — Slice-plane index
positive integer

Slice-plane index, specified as a positive integer. This index selects which slice to use to create the two-dimensional output.

Data Types: `double`

Output

Port_1 — Output
complex-valued matrix

Collapsed data cube, returned as a complex-valued matrix.

- If you set the **Output Slice** parameter to `Angle-Doppler`, the slice is the constant-range or fast-time plane. The output has dimensions Q-by-N.
- If you set the **Output Slice** parameter to `Range-Doppler`, the slice is the constant-angle plane. The output has dimension M-by-N.
- If you set the **Output Slice** parameter to `Range-angle`, the slice is the constant-Doppler plane. The output has dimension M-by-Q.

Data Types: `double`
Complex Number Support: Yes

Parameters

Output Slice — Output slice

`Angle-Doppler` (default) | `Range-Doppler` | `Range-angle`

Output slice, specified as `Angle-Doppler`, `Range-Doppler`, or `Range-angle`. The output slices are:

Slice plane	Slice dimensions
`Angle-Doppler`	*Q*-by-*N*
`Range-Doppler`	*M*-by-*N*
`Range-angle`	*M*-by-*Q*

Version History

Introduced in R2014b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

Dechirp Mixer

Dechirping operation on input signal

Libraries:
Phased Array System Toolbox / Detection

Description

The Dechirp block mixes the incoming signal from the `X` port with a reference signal incoming through the `RefX` port. The signals can be complex baseband signals. The input signal can be a matrix where each column is an independent channel. The reference signal is a vector. The reference signal is complex-conjugated and then multiplied with each signal column to compute the output.

Ports

Input

RefX — Reference signal
complex-valued *M*-by-1 vector

Reference signal, specified as a complex-valued *M*-by-1 vector.

Data Types: `single` | `double`
Complex Number Support: Yes

X — Input signal
complex-valued *M*-by-*N* matrix

Input signal, specified as a complex-valued *M*-by-*N* matrix. Each column of `X` is an independent signal and is individually mixed with the signal input to the `RefX` port.

Data Types: `single` | `double`
Complex Number Support: Yes

Output

Port_1 — Dechirped output signal
M-by-*N* matrix

Dechirped signal, returned as an *M*-by-*N* matrix. Each column of the `Port_1` port is the mixed output for the corresponding column of `X`.

Data Types: `single` | `double`
Complex Number Support: Yes

Version History

Introduced in R2014b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

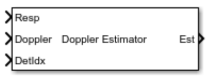
Usage notes and limitations:

Does not support variable-size inputs.

See Also
`dechirp`

Doppler Estimator

Doppler estimation



Libraries:
Phased Array System Toolbox / Detection

Description

The Doppler Estimator block estimates the Doppler (radial speed) of target detections obtained from the radar response data.

Ports

Input

Resp — Doppler-processed response data cube
complex-valued P-by-1 column vector | complex-valued M-by-P matrix | complex-valued M-by-N-by-P array

Doppler-processed response data cube, specified as a complex-valued P-by-1 column vector, a complex-valued M-by-P matrix, or a complex-valued M-by-N-by-P array.M represents the number of range samples, N is the number of sensor elements or beams, and P is the number of Doppler bins.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Doppler — Doppler grid values along Doppler dimension
real-valued P-by-1 column vector

Doppler grid values along the Doppler dimension, specified as a real-valued P-by-1 column vector. `Doppler` defines the Doppler values corresponding to the Doppler dimension of data input to the **Resp** port. Doppler values must be monotonically increasing and equally spaced. Units are in hertz or meters/sec.

Example: `[-0.3,-0.2,-0.1,0,0.1,0.2,0.3]`

Data Types: `double`

DetIdx — Detection indices
real-valued N_d-by-Q matrix

Detection indices, specified as a real-valued N_d-by-Q matrix. Q is the number of detections and N_d is the number of dimensions in the response data cube, **Resp**. Each column of **DetIdx** contains the indices of a detection in the response data cube.

NoisePower — Noise power at detection locations
positive scalar | real-valued 1-by-Q row vector of positive values

Noise power at detection locations, specified as a positive scalar or real-valued 1-by-*Q* row vector positive values. *Q* is the number of detections specified in the **DetIdx** input port.

Dependencies

To enable this port, select the **Output variance for parameter estimates** parameter, and then set **Source of noise power parameter** to `Input port`.

Clusters — Cluster IDs
real-valued 1-by-*Q* row vector of positive values

Cluster IDs, specified as a real-valued 1-by-*Q* row vector, where *Q* is the number of detections specified in the **DetIdx** input port. Each element of **Clusters** corresponds to an element of **DetIdx**.

Dependencies

To enable this input port, select the **Enable cluster ID input** checkbox.

Output

Est — Doppler estimate
real-valued *K*-by-1 column vector

Doppler estimates, returned as a real-valued *K*-by-1 column vector.

- When **Enable cluster ID input** is not selected, each Doppler estimate corresponds to one of the columns in the **DetIdx** input port. Then *K* equals the number of elements, *Q*, of **DetIdx**.
- When **Enable cluster ID input** is selected, each Doppler estimate corresponds to one of the cluster IDs in the **Clusters**input port. Then *K* equals the number of unique cluster IDs.

Var — Doppler estimation variance
positive, real-valued *K*-by-1 column vector

Doppler estimation variance, returned as a positive, real-valued *K*-by-1 column vector, where *K* is the dimension of **Est**. Each element of **Var** corresponds to an element of **Est**. The estimator variance is computed using the Ziv-Zakai bound.

Dependencies

To enable this port, select the **Output variance for parameter estimates** parameter.

Parameters

Maximum number of estimates — Maximum number of estimates to report
`1` (default) | positive integer

The maximum number of estimates to report, specified as a positive integer. When the number of requested estimates is greater than the number elements in **DetIdx**, the remainder is filled with `NaN`.

Data Types: `double`

Enable cluster ID input — Enable cluster ID input
`off` (default) | `on`

Enable the **Cluster** input port to pass in cluster association information.

Data Types: `Boolean`

Output variance for parameter estimates — Enable output variance port
`off` (default) | `on`

Enables the output of the parameter estimate variances via the **Var** port.

Data Types: `Boolean`

Number of pulses in Doppler processed waveform — Number of pulses
2 (default) | positive integer

Number of pulses in Doppler processed waveform, specified as a positive integer.

Dependencies

To enable this parameter, select the **Output variance for parameter estimates** `Output variance for parameter estimates` parameter.

Data Types: `double`

Source of noise power — Source of noise power values
`Property` (default) | `Input port`

Source of the noise power, specified as `Property` or `Input port`. If you set this parameter to `Property`, use the **Noise power** parameter to set the noise power at the detection locations. When set the parameter to `Input port`, specify noise power via the `NoisePower` input port.

Noise power — Noise power values
`1.0` (default) | positive scalar

Noise power for detections, specified as a positive scalar. The same noise power value is applied to all detections. Noise power is in linear units.

Dependencies

To enable this parameter, select the **Output variance for parameter estimates** checkbox and set the **Source of noise power** parameter to `Property`.

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2017a

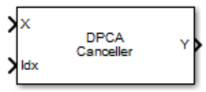
See Also

Blocks
CFAR Detector | 2-D CFAR Detector | Range Doppler Response

Objects
`phased.CFARDetector` | `phased.CFARDetector2D` | `phased.DopplerEstimator` | `phased.RangeDopplerResponse`

DPCA Canceller

Displaced phase center array (DPCA) pulse canceller for a uniform linear array



Libraries:
Phased Array System Toolbox / Space-Time Adaptive Processing

Description

The DPCA Canceller block filters clutter impinging on a uniform linear array using a displaced phase center array pulse canceller.

Ports

Input

X — Input signal
M-by-*N*-by-*P* complex-valued matrix

Input signal, specified as an *M*-by-*N*-by-*P* complex-valued array. *M* is the number of range samples, *N* is the number of channels, and *P* is the number of pulses.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Idx — Index of range cells
positive integer

Index of range cells to compute processing weights.

Example: `1`

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, specified as a positive scalar.

Dependencies

To enable this port, set the **Specify PRF as** parameter to `Input port`.

Data Types: `double`

Ang — Targeting direction
2-by-1 real-valued vector

Targeting direction, specified as a 2-by-*1* real-valued vector. The vector takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie

between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Specify direction as** parameter to `Input port`.

Data Types: `double`

Dop — Targeting Doppler frequency
scalar

Targeting Doppler frequency of current pulse, specified as a scalar.

Dependencies

This port appears when the **Output pre-Doppler result** check box is cleared and the **Specify targeting Doppler as** parameter is set to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-1 complex-valued vector

Processing output, returned as an *M*-by-*1* complex-valued vector. The quantity *M* is the number of range samples in the input port X.

Data Types: `double`

W — Processing weights
length *N*P* complex-valued vector

Processing weights, returned as Length *N*P* complex-valued vector. The quantity *N* is the number of channels and *P* is the number of pulses. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays. *L* is the number of desired beamforming directions specified in the Ang input port or by the **Beamforming direction (deg)** parameter. There is one set of weights for each beamforming direction.

Dependencies

To enable this port, select the **Enable weights output** check box.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Specify PRF as — Source of PRF value
`Property` (default) | `Input port`

Source of PRF value, specified as `Property` or `Input port`. When set to `Property`, the **Pulse repetition frequency (Hz)** parameter sets the PRF. When set to `Input port`, pass in the PRF using the PRF input port.

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1` (default) | positive scalar

Pulse repetition frequency, PRF, specified as a positive scalar. Units are in Hertz. Set this parameter to the same value set in any `Waveform` library block used in the simulation.

Dependencies

To enable this parameter, set the **Specify PRF as** parameter to `Property`.

Specify direction as — Specify source of targeting directions
`Property` (default) | `Input port`

Specify whether the targeting direction for the STAP processor block comes from a block parameter or from the ANG input port. Values of this parameter are

`Property`	• For the ADPCA Canceller and DPCA Canceller blocks, targeting direction is specified using **Receiving mainlobe direction (deg)**. • For the SMI Beamformer block, targeting direction is specified using **Targeting direction**. These parameters appear only when the **Specify direction as** parameter is set to `Property`.
`Input port`	Enter the targeting directions using the Ang input port. This port appears only when **Specify direction as** is set to `Input port`.

Receiving mainlobe direction (deg) — Pointing direction of main lobe of array
`[0;0]` (default) | real-valued 2-by-1 vector

Specify the direction of the main lobe of the receiving sensor array as a real-valued 2-by-1 vector. The direction is specified in the format of `[AzimuthAngle; ElevationAngle]`. The azimuth angle should be between –180° and 180° and the elevation angle should be between –90° and 90°.

Example: `[100;-45]`

Dependencies

To enable this parameter, set **Specify direction as** to `Property`.

Number of bits in phase shifters — Number of phase shift quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Specify targeting Doppler as — Source of targeting Doppler
`Property` (default) | `Input port`

Specify whether targeting Doppler values for the STAP processor comes from the **Targeting Doppler (Hz)** parameter of this block or using the `DOP` input port. For the ADPCA Canceller and DPCA Canceller blocks, the **Specify targeting Doppler as** parameter appears only when the **Output pre-Doppler result** check box is cleared. Values of this parameter are

`Property`	Specify targeting Doppler values using the **Targeting Doppler** parameter of the block. The **Targeting Doppler** parameter appears only when **Specify targeting Doppler as** is set to `Property`.
`Input port`	Specify targeting Doppler values using the `Dop` input port. This port appears only when **Specify targeting Doppler as** is set to `Input port`.

Targeting Doppler (Hz) — Targeting Doppler of STAP processor
`0` (default) | scalar

Targeting Doppler of STAP processor, specified as a scalar.

Dependencies

- To enable this parameter for the SMI Beamformer block, set **Specify targeting Doppler as** to `Property`.
- To enable this parameter for the ADPCA Canceller and DPCA Canceller blocks, first clear the **Output pre-Doppler result** check box. Then set the **Specify targeting Doppler as** parameter to `Property`.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Output pre-Doppler result — Output results before Doppler filtering
on (default) | off

Select this check box to output the results before Doppler filtering. Clear this check box to output the processing result after Doppler filtering. Selecting this check box will remove the **Specify targeting Doppler as** and **Targeting Doppler (Hz)** parameters.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Specify sensor array as — Type of array
`Array (no subarrays)` (default) | `MATLAB expression`

Specify a ULA sensor array directly or by using a MATLAB expression.

Types

`Array (no subarrays)`
`MATLAB expression`

Number of elements — Number of array elements in U
2 (default) | positive integer greater than or equal to two

The number of array elements for ULA arrays, specified as an integer greater than or equal to two.

Example: `11`

Data Types: `double`

Element spacing — Distance between ULA elements
`0.5` (default) | positive scalar

Distance between adjacent ULA elements, specified as a positive scalar. Units are in meters.

Example: `1.5`

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Taper — ULA array taper
`1` (default) | complex-valued vector

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Example: [0.5;1;0.5]

Data Types: `double`

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create a ULA array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.ULA('NumElements',13)`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Version History

Introduced in R2014b

See Also

`phased.DPCACanceller`

ESPRIT DOA

ESPRIT direction of arrival (DOA) estimator for ULA

Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The ESPRIT DOA block estimates the direction of arrival of a specified number of narrowband signals incident on a uniform linear array using the ESPRIT algorithm.

Ports

Input

Port_1 — Input signal
N-by-*M* complex-valued matrix

Input signal, specified as an *N*-by-*M* complex-valued matrix. *N* corresponds to the number of samples. *M* corresponds to the number data channels.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Output

Ang — Estimated broadside direction-of-arrival angles
M-by-1 real-valued vector

Estimated broadside direction-of-arrival angles, returned as an *M*-by-1 real-valued vector. Units are in degrees.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Number of signals — Number of signals

`1` (default) | positive integer scalar

Specify the number of signals as a positive integer scalar.

Spatial smoothing — Enable spatial smoothing
`0` (default) | non-negative integer

Specify the amount of averaging used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each increase in smoothing handles one extra coherent source, but reduces the effective number of elements by one. The maximum value of this parameter is $N - 2$, where N is the number of sensors in the ULA.

Type of least squares method — Type of least squares method

`TLS` (default) | `LS`

Specify the least squares method used for ESPRIT as one of `TLS` or `LS` where `TLS` refers to total least squares and `LS`refers to least squares.

Forward-backward averaging — Enable forward-backward averaging
off (default) | on

Select this parameter to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold structure.

Row weighting factor — Row weighting factor

`1` (default)

Specify the row weighting factor for signal subspace eigenvectors as a positive integer scalar. This parameter controls the weights applied to the selection matrices. In most cases higher value are better. However, the value can never be greater than $(N-1)/2$ where N is the number of elements of the array.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cardioid Antenna` | `Cosine Antenna` | `Custom Antenna` | `Gaussian Antenna` | `Sinc Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cardioid Antenna`
- `Cosine Antenna`

- `Custom Antenna`
- `Gaussian Antenna`
- `Sinc Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Null axis direction — Null axis direction
`-x` (default) | `+x` | `+y` | `-y` | `+z` | `-z`

Dependencies

To enable this parameter, set **Element type** to `Cardioid Antenna`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-L row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Align element normal with array normal — Align element normal with array normal
on (default) | off

Dependencies

This parameter is enabled when **Element type** is set to `Custom Antenna`.

Radiation pattern beamwidth (deg) — Radiation pattern beamwidth
`[10,10]` (default)

Dependencies

This parameter is enabled when **Element type** is set to `Gaussian Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

ULA Parameters

Number of elements — Number of array elements in U
`2` (default) | positive integer greater than or equal to two

The number of array elements for ULA arrays, specified as an integer greater than or equal to two.

Example: `11`

Data Types: `double`

Element spacing — Distance between ULA elements
`0.5` (default) | positive scalar

Distance between adjacent ULA elements, specified as a positive scalar. Units are in meters.

Example: `1.5`

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Taper — ULA array taper
`1` (default) | complex-valued vector

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Example: [0.5;1;0.5]

Data Types: `double`

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create a ULA array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.ULA('NumElements',13)`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Version History

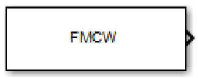
Introduced in R2014b

See Also

`phased.ESPRITEstimator`

FMCW Waveform

Frequency-modulated continuous (FMCW) waveform source



Libraries:
Phased Array System Toolbox / Waveforms

Description

The FMCW Waveform block generates a frequency modulated continuous wave (FMCW) waveform with a specified sweep time and sweep bandwidth. The block output can be either an integer number of pulses or samples.

Ports

Output

Port_1 — FMCW Waveform
complex-valued column vector

FMCW waveform, returned as a complex-valued column vector containing the waveform samples.

- If the **Output signal format** parameter is set to `Samples`, then the output of the port consists of the number of samples set by the **Number of samples in output** parameter.
- If the **Output signal format** parameter is set to `Sweeps`, then the output of the port consists of the number of sweeps set by the **Number of sweeps in output** parameter. Also, if the **Sweep direction** parameter is set to `Triangle` each sweep is one-half period in duration.

Parameters

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.

Programmatic Use
Block Parameter:`SampleRate`
Type:double
Values:positive scalar
Default:`1e6`

Sweep time (s) — Duration of each linear FM sweep

`1e-4` (default)

Specify the duration, in seconds, of the upsweep or the downsweep of the signal as a scalar or row vector of positive, real numbers. The product of the **Sample rate** value and each **Sweep time** entry must be an integer.

To implement a varying sweep time, specify **Sweep time** as a row vector. The waveform uses successive entries of the vector as the sweep time for successive periods of the waveform. If the last element of the vector is reached, the process continues cyclically with the first entry of the vector.

If **Sweep time** and **Sweep bandwidth** are both row vectors, the vectors must have the same length.

If **Sweep direction** is `Up` or `Down`, the sweep period equals the sweep time. If **Sweep direction** is `Triangle`, the sweep period is twice the sweep time because each period consists of an upsweep segment and a downsweep segment.

Sweep bandwidth (Hz) — FM sweep bandwidth

`1e5` (default)

Specify the bandwidth of the linear FM sweeping, in hertz, as a scalar or row vector of positive, real numbers.

To implement a varying bandwidth, specify **Sweep bandwidth** as a row vector. The waveform uses successive entries of the vector as the sweep bandwidth for successive periods of the waveform. If the waveform reaches the last element of the **Sweep bandwidth** vector, the process continues cyclically with the first entry of the vector.

If **Sweep time** and **Sweep bandwidth** are both row vectors, the vectors must have the same length.

Sweep direction — FM sweep direction

`Up` (default) | `Down` | `Triangle`

Specify the direction of the linear FM sweep as one of `Up`, `Down`, or `Triangle`.

Sweep interval — Location of FM sweep interval

`Positive` (default) | `Symmetric`

If you set this parameter value to `Positive`, the waveform sweeps in the interval between 0 and *B*, where *B* is the value of the **Sweep bandwidth** parameter. If you set this parameter to `Symmetric`, the waveform sweeps in the interval between –*B/2* and *B/2*.

Output signal format — Output signal format

`Sweeps` (default) | `Samples`

Specify the format of the output signal as `Sweeps` or `Samples`.

If you set this parameter to `Sweeps`, the output of the block is in the form of multiple sweeps. The number of sweeps is the value of the **Number of sweeps in output** parameter.

If you set this parameter to `Samples`, the output of the block is in the form of multiple samples. The number of samples is the value of the **Number of samples in output** parameter.

If the **Sweep direction** parameter is set to `Triangle`, each sweep is one-half of a period.

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use
Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Number of sweeps in output — Number of sweeps in output

`1` (default)

Specify the number of sweeps in the block output as a positive integer. This parameter appears only when you set **Output signal format** to `Sweeps`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also
`phased.FMCWWaveform`

Free Space

Free space environment

Libraries:
Phased Array System Toolbox / Environment and Target

Description

The Free Space Channel block propagates a signal from one point to another in space. The block models propagation time, free space propagation loss and Doppler shift. The block assumes that the propagation speed is much greater than the target or array speed in which case the stop-and-hop model is valid.

When propagating a signal in free-space to an object and back, you have the choice of either using a single block to compute a two-way free space propagation delay or two blocks to perform one-way propagation delays in each direction. Because the free-space propagation delay is not necessarily an integer multiple of the sampling interval, it may turn out that the total round trip delay in samples when you use a two-way propagation block differs from the delay in samples when you use two one-way propagation blocks. For this reason, it is recommended that, when possible, you use a single two-way propagation block.

Ports

Input

X — Narrowband signal
M-element complex-valued column vector, *M*-by-*N* complex-valued matrix.

Narrowband signal, specified as an *M*-element complex-valued column vector or *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values of the signal and *N* is the number of signals to propagate. When you specify *N* signals, you need to specify *N* signal origins or *N* signal destinations.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Pos1 — Signal origin
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Signal origin, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of propagated signals and equals the dimension specified in the signal into the port X. If `Pos1` is a column vector, it takes the form `[x; y; z]`. If `Pos1` is a matrix, each column specifies a different signal origin and has the form `[x; y; z]`. `Pos1` and `Pos2` cannot both be specified as matrices — at least one must be a 3-by-1 column vector. Position units are meters.

Data Types: `double`

Pos2 — Signal destination
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Signal destination, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of propagated signals and equals the dimension specified in the signal into the port X. If `Pos2` is a column vector, it takes the form `[x; y; z]`. If `Pos2` is a matrix, each column specifies a different signal origin and has the form `[x; y; z]`. `Pos2` and `Pos1` cannot both be specified as matrices — at least one must be a 3-by-1 column vector. Position units are meters.

Data Types: `double`

Vel1 — Signal origin velocity
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Signal origin velocity, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of propagated signals and equals the dimension specified in the signal into the port X. If `Vel1` is a column vector, it takes the form `[Vx; Vy; zV]`. If `Vel1` is a matrix, each column specifies a different signal origin and has the form `[Vx; Vy; Vz]`. `Vel1` and `Vel2` cannot both be specified as matrices — at least one must be a 3-by-1 column vector. Position units are meters.

Data Types: `double`

Vel2 — Signal destination velocity
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Signal destination velocity, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of propagated signals and equals the dimension specified in the signal into the port X. If `Vel2` is a column vector, it takes the form `[Vx; Vy; zV]`. If `Vel2` is a matrix, each column specifies a different signal origin and has the form `[Vx; Vy; Vz]`. `Vel2` and `Vel1` cannot both be specified as matrices — at least one must be a 3-by-1 column vector. Position units are meters.

Data Types: `double`

Output

Port_1 — Propagated narrowband signal
M-element complex-valued column vector, *M*-by-*N* complex-valued matrix.

Propagated signal, returned as a *M*-element complex-valued column vector, *M*-by-*N* complex-valued matrix.

If X is a column vector or matrix, Y is also a column vector or matrix with the same dimensions.

The output Y contains signal samples arriving at the signal destination within the current time frame. The current time frame is defined as the time spanned by the current input. Whenever it takes longer than the current time frame for the signal to propagate from the origin to the destination, the output contains no contribution from the input of the current time frame.

Parameters

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`.

Data Types: `double`

Signal carrier frequency (Hz) — Signal carrier frequency
`3e8` (default) | positive real-valued scalar

Signal carrier frequency, specified as a positive real-valued scalar. Units are in hertz.

Data Types: `double`

Perform two-way propagation — Perform two-way propagation

off (default) | on

Select this check box to perform round-trip propagation between the origin and destination. Otherwise the block performs one-way propagation from the origin to the destination.

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Maximum one-way propagation distance (m) — Maximum one-way propagation distance

`10e3` (default)

The maximum distance, in meters, between the origin and the destination as a positive scalar. Amplitudes of any signals that propagate beyond this distance will be set to zero.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Algorithms

When the origin and destination are stationary relative to each other, the block output can be written as $y(t) = x(t - \tau)/L$. The quantity τ is the delay and L is the propagation loss. The delay is computed from $\tau = R/c$ where R is the propagation distance and c is the propagation speed. The free space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far-field of the transmitting element or array. In the near-field, the free-space path loss formula is not valid and can result in losses smaller than one, equivalent to a signal gain. For this reason, the loss is set to unity for range values, $R \le \lambda/4\pi$.

When there is relative motion between the origin and destination, the processing also introduces a frequency shift. This shift corresponds to the Doppler shift between the origin and destination. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The parameter v is the relative speed of the destination with respect to the origin.

Version History

Introduced in R2014b

See Also

Wideband Free Space | `phased.FreeSpace`

Frost Beamformer

Frost beamformer

Libraries:
Phased Array System Toolbox / Beamforming

Description

The Frost Beamformer block implements a Frost beamformer. A Frost beamformer consists of a time-domain MVDR beamformer followed by a bank of FIR filters. The MVDR beamformer steers the beam towards a given direction while the FIR filters preserve the input signal power.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

XT — Training signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this port, select the **Enable training data input** check box.

Data Types: `double`

Ang — Beamforming direction
2-by-1 real-valued vector

Beamforming direction, specified as a 2-by-1 real-valued vector taking the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Source of beamforming direction** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-1 complex-valued vector

Beamformed output, returned as an *M*-by-1 complex-valued vector. The quantity *M* is the number of signal samples.

Data Types: `double`

W — Beamforming weights
N-by-1 complex-valued vector

Beamformed weights, returned as an *N*-by-1 complex-valued vector. The quantity *N* is the number of array elements. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays.

Dependencies

To enable this port, select the **Enable weights output** check box.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

FIR filter length — FIR filter length
`1` (default) | positive integer

The length of the FIR filter used to process each sensor element data, specified as a positive integer.

Data Types: `double`

Diagonal loading factor — Diagonal loading factor for stability
nonnegative scalar

Specify the diagonal loading factor as a nonnegative scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample support is small.

Enable training data input — Enable the use of training data
off (default) | on

Select this check box to specify additional training data via the input port `XT`. To use the input signal as the training data, clear the check box which removes the port.

Source of beamforming direction — Source of beamforming direction
`Property` (default) | `Input port`

Source of beamforming direction, specified as `Property` or `Input port`. When you set **Source of beamforming direction** to `Property`, you then set the direction using the **Beamforming direction (deg)** parameter. When you select `Input port`, the direction is determined by the input to the `Ang` port.

Beamforming direction (deg) — Beamforming direction
2-by-1 real-valued vector

Beamforming direction, specified as a 2-by-1 real-valued vector taking the form `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this parameter, set the **Source of beamforming direction** parameter to `Property`.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><td rowspan="2">Block Simulation</td><td colspan="3">Simulation Behavior</td></tr>
<tr><td><code>Normal</code></td><td><code>Accelerator</code></td><td><code>Rapid Accelerator</code></td></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.

- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
ULA (default) | URA | UCA | `Conformal Array`

Array geometry, specified as one of

- ULA — Uniform linear array
- URA — Uniform rectangular array

- UCA — Uniform circular array
- Conformal Array — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

Dependencies

To enable this parameter, set **Geometry** to ULA or UCA.

Element spacing (m) — Spacing between array elements
0.5 for ULA arrays and [0.5,0.5] for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form [SpacingBetweenArrayRows,SpacingBetweenArrayColumns].

Dependencies

To enable this parameter, set **Geometry** to ULA or URA.

Array axis — Linear axis direction of ULA
y (default) | x | z

Linear axis direction of ULA, specified as y, x, or z. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

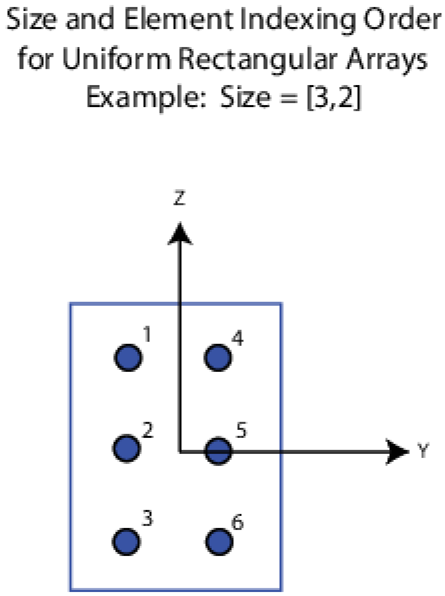
- To enable this parameter, set **Geometry** to ULA.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form [NumberOfArrayRows,NumberOfArrayColumns].
- If **Array size** is an integer, the array has the same number of elements in each row and column.

For a URA, array elements are indexed from top to bottom along the leftmost array column, and continued to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Data Types: `double`

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. If the parameter value is a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

To enable this parameter, set **Geometry** to `Conformal Array`.

Data Types: `double`

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Version History

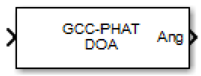
Introduced in R2014b

See Also

`phased.FrostBeamformer`

GCC DOA and TOA

Generalized cross-correlator with phase transform



Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The GCC DOA and TOA block estimates the direction of arrival and time of arrival of a signal at an array. The block uses a generalized cross-correlation with phase transform *(GCC-PHAT)* algorithm.

Ports

Input

Port_1 — Received signal
M-by-*N* complex-valued matrix

Received signal, input as an *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values (snapshots) of the signal and *N* is the number of sensor elements in the array. For subarrays, *N* is the number of subarrays.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`
Complex Number Support: Yes

Output

Ang — Direction of arrival
2-by-1 real-valued column vector | scalar

Direction of arrival of a signal, output as a 2-by-1 real-valued column vector in the form `[azimuth;elevation]`. If the array is a uniform linear array, `Ang` is a scalar representing the broadside angle. Angle units are in degrees defined with respect to the local coordinate system of the array.

Data Types: `double`

Tau — Time delays of arrival
1-by-*P* real-valued row vector

Time delays of arrival, output as 1-by-*P* real-valued row vector. *P* is the number of sensor pairs selected from the array.

- When **Source of sensor pairs** is set to `Auto`, *P* = *N* - *1*. *N* is the number of elements in the array.
- When **Source of sensor pairs** is set to `Property`, *P* is the number of sensor pairs specified by the **Sensor pairs** parameter.

Time units are seconds.

Dependencies

To enable this port, select the **Enable delay output** check box.

Data Types: `double`

Rxy — Estimated cross-correlation
(2*M*+1)-by-*P* complex-valued matrix

Estimated cross-correlation between pairs of sensors, output as a (2*M*+1)-by-*P* complex-valued matrix, where *P* is the number of sensor pairs selected from the array. *M* is the number of time samples in the input signal.

- When **Source of sensor pairs** is set to `Auto`, $P = N - 1$. *N* is the number of elements in the array. The columns in `Rxy` contain the correlations between the first sensor and all other sensors.
- When **Source of sensor pairs** is set to `Property`, *P* is the number of sensor pairs specified by the **Sensor pairs** parameter. Each column in `Rxy` contains the correlation for the corresponding pair of sensors.

Dependencies

To enable this port, select the **Enable correlation output** check box.

Data Types: `double`
Complex Number Support: Yes

Lag — Time lags
M-by-1 real-valued column vector

Time lags, output as an (2*M*+1)-by-1 real-valued column vector. *M* is the number of time samples in the input signal. Each time lag applies to the corresponding row in the cross-correlation matrix.

Dependencies

To enable this port, select the **Enable correlation output** check box.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Source of sensor pairs — Source of sensor pairs

`Auto` (default) | `Property`

Source

`Property`	When you set this parameter to `Property`, specify the sensor pairs for computing correlation using the **Sensor pairs** parameter.
`Auto`	When you set this parameter to `Auto`, correlations are computed between the first element and all other elements. The first element serves as the reference channel.

Sensor pairs — Sensor pairs

`[2;1]` (default)

Sensor pairs, specified as a 2-by-*M* matrix of strictly positive integers.

Dependencies

This parameter appears only when you set the **Source of sensor pairs** parameter to `Property`.

Enable correlation output — Option to enable correlation output

off (default) | on

Check this box to output the correlations computed using the GCC-PHAT algorithm as well as the corresponding lags between sensor pairs. Correlation values are output via the `Rxy` port. Lag values are output via the `Lags` port. These ports appear only when you check the **Enable correlation output** box. Clear this check box to disable output of correlations.

Enable delay output — Option to enable delay output

off (default) | on

Select this check box to output the delay corresponding to the arrival angle of a signal between each sensor pair. The delay is output in the `Tau` port. This port appears only when you check the **Enable delay output** box. Clear this check box to disable output of delays.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.

- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cardioid Antenna` | `Cosine Antenna` | `Custom Antenna` | `Gaussian Antenna` | `Sinc Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cardioid Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Gaussian Antenna`
- `Sinc Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Null axis direction — Null axis direction
`-x` (default) | `+x` | `+y` | `-y` | `+z` | `-z`

Dependencies

To enable this parameter, set **Element type** to `Cardioid Antenna`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-L row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Align element normal with array normal — Align element normal with array normal
on (default) | off

Dependencies

This parameter is enabled when **Element type** is set to `Custom Antenna`.

Radiation pattern beamwidth (deg) — Radiation pattern beamwidth
`[10,10]` (default)

Dependencies

This parameter is enabled when **Element type** is set to `Gaussian Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array

- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

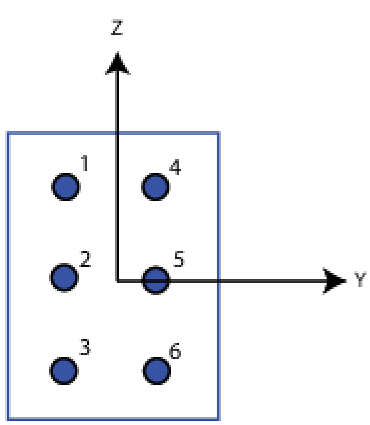
Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-N row vector. In this vector, N represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an M-by-N matrix. M is the number of subarrays and N is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

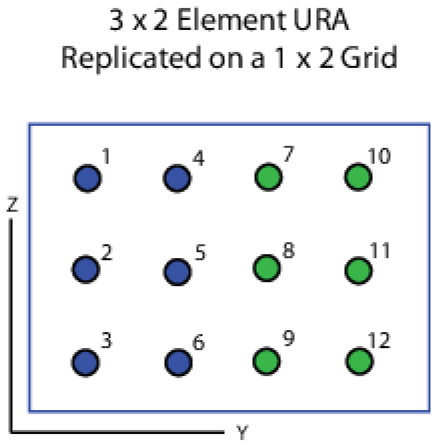
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Version History

Introduced in R2015b

See Also

`phased.GCCEstimator` | `gccphat`

GSC Beamformer

Generalized sidelobe canceller

Libraries:
Phased Array System Toolbox / Beamforming

Description

The GSC Beamformerblock implements a generalized sidelobe cancellation (GSC) beamformer. A GSC beamformer splits an arrays incoming signals and sends them through a conventional beamformer path and a sidelobe canceling path. The algorithm first presteers the array to the beamforming direction and then adaptively chooses filter weights to minimize power at the output of the sidelobe canceling path. The algorithm uses least mean squares (LMS) to compute the adaptive weights. The final beamformed signal is the difference between the outputs of the two paths.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Ang — Beamforming direction
2-by-1 real-valued vector

Beamforming direction, specified as a *2*-by-1 real-valued vector, where taking the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Source of beamforming direction** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-1 complex-valued vector

Beamformed output, returned as an M-by-1 complex-valued vector. The quantity M is the number of signal samples.

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Signal path FIR filter length — Length of the FIR filter along the signal path
`1` (default) | positive integer

Length of the signal path FIR filter, specified as a positive integer. The FIR filter is a delta function.

Adaptive filter step size — LMS adaptive filter step size factor
`0.1` (default) | positive scalar

The adaptive filter step size factor, specified as a positive scalar. This quantity, when divided by the total power in the sidelobe canceling path, determines the actual adaptive filter step size used by the LMS algorithm.

Beamforming direction (deg) — Beamforming direction
2-by-1 real-valued vector

Beamforming direction, specified as a 2-by-1 real-valued vector taking the form `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this parameter, set the **Source of beamforming direction** parameter to `Property`.

Source of beamforming direction — Source of beamforming direction
`Property` (default) | `Input port`

Source of beamforming direction, specified as `Property` or `Input port`. When you set **Source of beamforming direction** to `Property`, you then set the direction using the **Beamforming direction (deg)** parameter. When you select `Input port`, the direction is determined by the input to the `Ang` port.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum

Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued Q-by-P matrix | real-valued Q-by-P-by-L array

Magnitude of the combined antenna radiation pattern, specified as a Q-by-P matrix or a Q-by-P-by-L array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, Q equals the length of the vector specified by the **Elevation angles (deg)** parameter and P equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, Q equals the length of the vector specified by the **Theta Angles (deg)** parameter and P equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity L equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued Q-by-P matrix | real-valued Q-by-P-by-L array

Phase of the combined antenna radiation pattern, specified as a Q-by-P matrix or a Q-by-P-by-L array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, Q equals the length of the vector specified by the **Elevation angles (deg)** parameter and P equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, Q equals the length of the vector specified by the **Theta Angles (deg)** parameter and P equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity L equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees

azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
ULA (default) | URA | UCA | `Conformal Array`

Array geometry, specified as one of

- ULA — Uniform linear array
- URA — Uniform rectangular array
- UCA — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

Dependencies

To enable this parameter, set **Geometry** to ULA or UCA.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.

Dependencies

To enable this parameter, set **Geometry** to ULA or URA.

Array axis — Linear axis direction of ULA
y (default) | x | z

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to ULA.
- This parameter is also enabled when the block only supports ULA arrays.

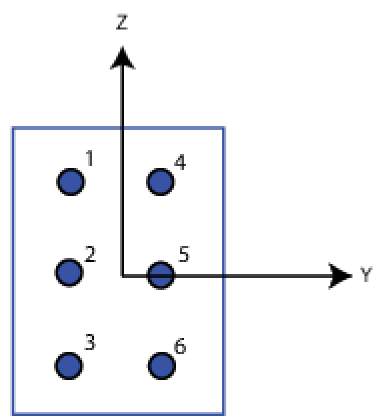
Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of elements in each row and column.

For a URA, array elements are indexed from top to bottom along the leftmost array column, and continued to the next columns from left to right. In this figure, the **Array size** value of [3,2] creates an array having three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Dependencies

To enable this parameter, set **Geometry** to URA.

Element lattice — Lattice of URA element positions
Rectangular (default) | Triangular

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to URA.

Array normal — Array normal direction
x for URA arrays or z for UCA arrays (default) | y

Array normal direction, specified as x, y, or z.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Data Types: `double`

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. If the parameter value is a matrix, each

column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

To enable this parameter, set **Geometry** to `Conformal Array`.

Data Types: `double`

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

More About

Generalized Sidelobe Cancellation

The generalized sidelobe canceller (GSC) is an efficient implementation of a linear constraint minimum variance (LCMV) beamformer. LCMV beamforming minimizes the output power of an array while preserving the power in one or more specified directions. This type of beamformer is called a *constrained beamformer*. You can compute exact weights for the constrained beamformer but the computation is costly when the number of elements is large. The computation requires the inversion of a large spatial covariance matrix. The GSC formulation converts the adaptive constrained optimization LCMV problem into an adaptive unconstrained problem, which simplifies the implementation.

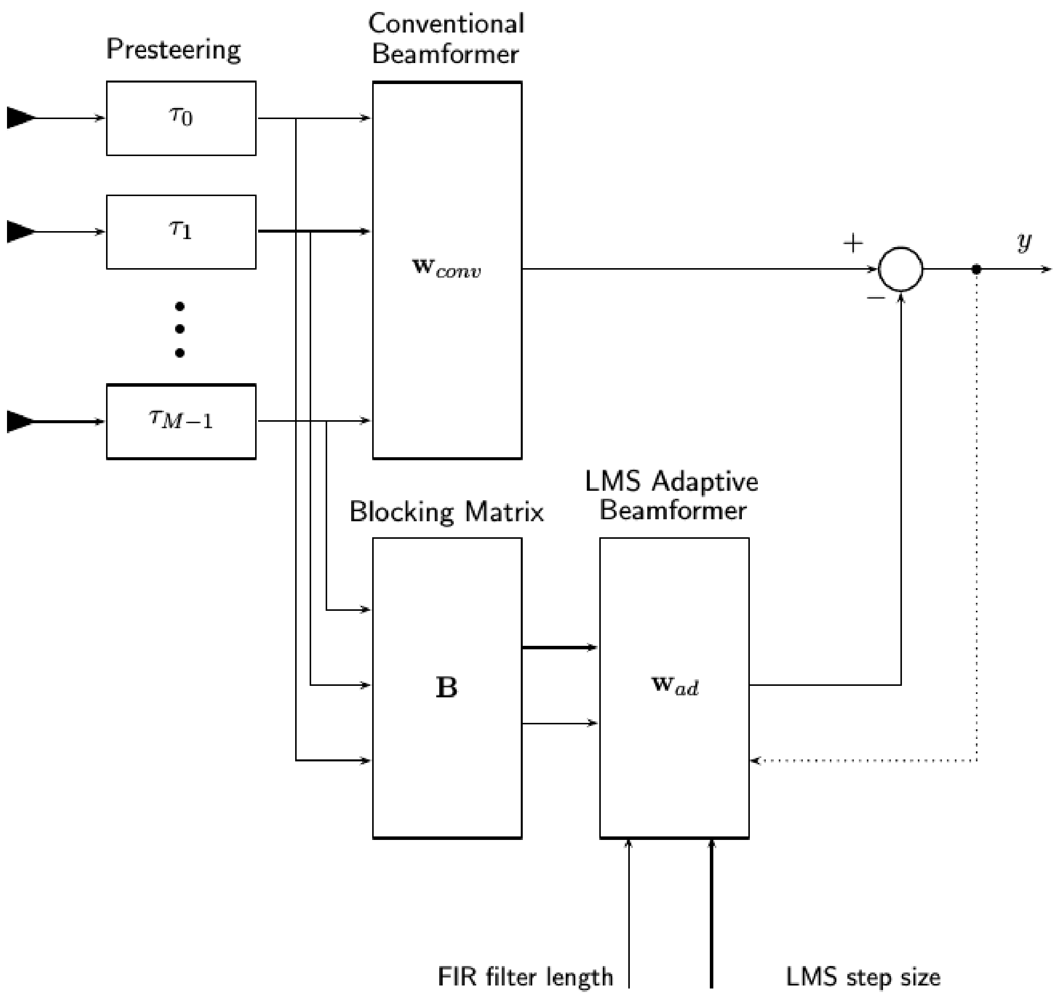
In the GSC algorithm, incoming sensor data is split into two signal paths as shown in the block diagram. The upper path is a conventional beamformer. The lower path is an adaptive unconstrained beamformer whose purpose is to minimize the GSC output power. The GSC algorithm consists of these steps:

1. Presteer the element sensor data by time-shifting the incoming signals. Presteering time-aligns all sensor element signals. The time shifts depend on the arrival angle of the signal.
2. Pass the presteered signals through the upper path into a conventional beamformer with fixed weights, $\mathbf{w}_{conv}$.
3. Also pass the presteered signals through the lower path into the blocking matrix, $\mathbf{B}$. The blocking matrix is orthogonal to the signal and removes the signal from the lower path.

4 Filter the lower path signals through a bank of FIR filters. The `FilterLength` property sets the length of the filters. The filter coefficients are the adaptive filter weights, $\mathbf{w}_{ad}$.

5 Compute the difference between the upper and lower signal paths. This difference is the beamformed GSC output.

6 Feed the beamformed output back into the filter. Adapt the filter weights using a least mean-square (LMS) algorithm. The adaptive LMS step size is the quantity set by the `LMSStepSizeFactor` property, divided by the total signal power.



Version History

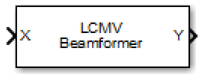
Introduced in R2016b

See Also

`phased.GSCBeamformer`

LCMV Beamformer

Narrowband linear constraint minimum variance (LCMV) beamformer



Libraries:
Phased Array System Toolbox / Beamforming

Description

The LCMV Beamformer block performs narrowband linear-constraint minimum-variance (LCMV) beamforming. The number of constraints must be less than the number of elements or subarrays in the array.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signals to beamformer, specified as an *M*-by-*N* complex-valued matrix. *M* is the number of signal samples. *N* is the number of sensor array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

XT — Training signal
P-by-*N* complex-valued matrix

Training input signal, specified as a *P*-by-*N* complex-valued matrix. *P* is the number of samples in the training input signal. *N* is the number of elements of the sensor array. *P* must be greater than *N*.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this port, select the **Enable training data input** checkbox.

Data Types: `double`

Output

Y — Beamformed output
M-by-1 complex-valued column vector

Beamformed output, specified as an M-by-1 complex-valued column vector. M is the number of signal samples.

Data Types: `double`

W — Beamformer weights output
N-by-1 complex-valued column vector

Beamformer weights output, specified as an N-by-1 complex-valued column vector. N is the number of array elements.

Dependencies

To enable this port, select the **Enable weights output** checkbox.

Data Types: `double`

Parameters

Constraint matrix — LCMV beamformer constraint matrix
`complex([1;1])` (default) | N-by-K complex-valued matrix

LCMV beamformer constraint matrix specified as an N-by-K complex-valued matrix. Each column of the matrix is a constraint. N is the number of elements in the sensor array and K is the number of constraints. K must be less than or equal to the number of sensors, $N,K \leq N$

Desired response vector — Desired response for LCMV beamforming
`1` (default) | real-valued K-by-1 column vector

Desired response of the LCMV beamformer, specified as a real-valued K-by-1 column vector. K is the number of constraints in the **Constraint matrix**. Each element in the vector defines the desired response of the constraint specified in the corresponding column of the **Constraint matrix** parameter.

Diagonal loading factor — Diagonal loading factor
positive scalar

Diagonal loading factor, specified as a positive scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample support is small.

Enable training data input — Enable training data input port
`off` (default) | `on`

Enable training data input port, specified as `off` or `on`. To enable the training data input port, `XT`, select this checkbox.

Enable weights output — Enable output of beamformer weights
`off` (default) | `on`

Enable beamforming weights output port, specified as `off` or `on`. To enable the beamforming weights output port, `W`, select this checkbox.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also
`phased.LCMVBeamformer`

Linear FM Waveform

Linear FM (LFM) pulse waveform

Libraries:
Phased Array System Toolbox / Waveforms

Description

The Linear FM Waveform block generates a linear FM pulse waveform with specified pulse width, pulse repetition frequency (PRF), and sweep bandwidth. The block outputs an integer number of pulses or samples.

Ports

Input

PRFIdx — PRF Index
positive integer

Index to select the pulse repetition frequency (PRF), specified as a positive integer. The index selects the PRF from the predefined vector of values specified by the **Pulse repetition frequency (Hz)** parameter.

Example: 4

Dependencies

To enable this port, select **Enable PRF selection input**.

Data Types: `double`

FreqOffset — Frequency offset
scalar

Frequency offset in Hz, specified as a scalar.

Example: `2e3`

Dependencies

To enable this port, set **Source of Frequency Offset** to `Input port`.

Data Types: `double`

Output

Y — Pulse waveform
complex-valued vector

Pulse waveform samples, returned as a complex-valued vector.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, returned as a positive scalar.

Dependencies

To enable this port, set the **Output signal format** parameter to `Pulses` and then select the **Enable PRF output** parameter.

Data Types: `double`

Coeff — Matched filter coefficients
vector | matrix

Matched filter coefficients, returned as a vector or matrix.

Dependencies

To enable this port, select **Enable Matched Filter Coefficients Output**.

Data Types: `double`

Parameters

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.

Programmatic Use
Block Parameter:`SampleRate`
Type:double
Values:positive scalar
Default:`1e6`

Method to specify pulse duration — Pulse duration as time or duty cycle
`Pulse width` (default) | `Duty cycle`

Method to set the pulse duration, specified as `Pulse width` or `Duty cycle`. When you set this parameter to `Pulse width`, the pulse duration is set using the **Pulse width (s)** parameter. When you set this parameter to `Duty cycle`, the pulse duration is computed from the values of the **Pulse repetition frequency (Hz)** and **Duty Cycle** parameters.

Programmatic Use
Block Parameter:`DurationSpecification`
Type:string
Values:string
Default:`'Pulse width'`

Pulse width (s) — Time duration of pulse
`50e-6` (default) | positive scalar

The duration of each pulse, specified as a positive scalar. Set the product of **Pulse width (s)** and **Pulse repetition frequency** to be less than or equal to one. This restriction ensures that the pulse width is smaller than the pulse repetition interval. Units are in seconds.

Example: `300e-6`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Pulse width`.

Programmatic Use
Block Parameter:`PulseWidth`
Type:double
Values:string
Default:`50e-6`

Duty cycle — Waveform duty cycle
`0.5` (default) | scalar in the range *[0,1]*

Waveform duty cycle, specified as a scalar in the range *[0,1]*.

Example: `0.7`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Duty cycle`.

Programmatic Use
Block Parameter:`DutyCycle`
Type:double
Values:positive scalar
Default:`1e6`

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1e4` (default) | positive scalar

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The value of **Pulse repetition frequency (Hz)** must satisfy these constraints:

- The product of **Pulse width** and **Pulse repetition frequency (Hz)** must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of **Pulse repetition frequency** must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* by using block parameter settings alone or in conjunction with the input port, `PRFIdx`.

- When the **Enable PRF selection input** parameter is not selected, set the *PRF* using block parameters.

- To implement a constant *PRF*, specify **Pulse repetition frequency (Hz)** as a positive scalar.
- To implement a staggered *PRF*, specify **Pulse repetition frequency (Hz)** as a row vector with positive values. After the waveform reaches the last element of the vector, the process continues cyclically with the first element of the vector. When *PRF* is staggered, the time between successive output pulses cycles through the successive values of the *PRF* vector.

- When the **Enable PRF selection input** parameter is selected, you can implement a selectable *PRF* by specifying **Pulse repetition frequency (Hz)** as a row vector with positive real-valued entries. But this time, when you execute the block, select a *PRF* by passing an index into the *PRF* vector into the `PRFIdx` port.

In all cases, the number of output samples is fixed when you set the **Output signal format** to `Samples`. When you use a varying *PRF* and set **Output signal format** to `Pulses`, the number of output samples can vary.

Programmatic Use
Block Parameter:PRF
Type:double
Values:positive scalar
Default:`1e6`

Enable PRF selection input — Select predefined PRF
off (default) | on

Select this parameter to enable the `PRFIdx` port.

- When enabled, pass in an index into a vector of predefined PRFs. Set predefined PRFs using the **Pulse repetition frequency (Hz)** parameter.
- When not enabled, the block cycles through the vector of PRFs specified by the **Pulse repetition frequency (Hz)** parameter. If **Pulse repetition frequency (Hz)** is a scalar, the PRF is constant.

Programmatic Use
Block Parameter:`PRFSelectionInputPort`
Type:logical
Values:positive scalar
Default:`off`

Sweep bandwidth (Hz) — Bandwidth of FM sweep
`1e5` (default) | positive scalar

Bandwidth of the linear FM sweep in Hz, specified as a positive scalar.

Example: `1e3`

Sweep direction — Direction of FM sweep
Up (default) | Down

Specify the direction of the linear FM sweep as Up (increasing frequency) or Down (decreasing frequency).

Sweep interval — FM frequency sweep interval
`Positive` (default) | `Symmetric`

FM frequency sweep interval, specified as `Positive` or `Symmetric`. If you set this parameter to `Positive`, the waveform sweeps the frequency interval between *0* and *B*, where *B* is the value of the **Sweep bandwidth** parameter. If you set this parameter value to `Symmetric`, the waveform sweeps the interval between *–B/2* and *B/2*.

Envelope function — FM signal amplitude envelope
`Rectangular` (default) | `Gaussian`

FM signal amplitude envelope, specified as `Rectangular` or `Gaussian`.

Source of Frequency Offset — Source of frequency offset
`Property` (default) | `Input port`

Source of frequency offset, specified as `Property` or `Input port`.

- When set to `Property`, the offset is determined by the value of the **Frequency Offset** parameter.
- When set to `Input port`, the offset is determined by the value of the **FreqOffset** port.

Programmatic Use
Block Parameter:`FrequencyOffsetSource`
Type:enum
Values:`Property`, `Input Port`
Default:`Property`

Frequency Offset (Hz) — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `2e3`

Dependencies

To enable this parameter set the **Source of Frequency Offset** parameter to `Input port`.

Programmatic Use
Block Parameter:`FrequencyOffset`
Type:double
Values:`scalar`
Default:`0`

Source of simulation sample time — Source of simulation sample time
`Derive from waveform parameters` (default) | `Inherit from Simulink engine`

Source of simulation sample time, specified as `Derive from waveform parameters` or `Inherit from Simulink engine`. When set to `Derive from waveform parameters`, the block runs at a variable rate determined by the PRF of the selected waveform. The elapsed time is variable. When set to `Inherit from Simulink engine`, the block runs at a fixed rate so the elapsed time is a constant.

Dependencies

To enable this parameter, select the **Enable PRF selection input** parameter.

Programmatic Use
Block Parameter:`SimulationTimeSource`
Type:enum
Values:`Derive from waveform parameters`, `Inherit from Simulink engine`
Default:`Derive from waveform parameters`

Output signal format — Format of the output signal
`Pulses` (default) | `Samples`

The format of the output signal, specified as `Pulses` or `Samples`.

If you set this parameter to `Samples`, the output of the block consists of multiple samples. The number of samples is the value of the **Number of samples in output** parameter.

If you set this parameter to `Pulses`, the output of the block consists of multiple pulses. The number of pulses is the value of the **Number of pulses in output** parameter.

Programmatic Use
Block Parameter:`OutputFormat`
Type:enum
Values:`Pulses` `Samples`
Default:`Pulses`

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use
Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Number of pulses in output — Number of pulses in output
`1` (default) | positive integer

Number of pulses in the block output, specified as a positive integer.

Example: 2

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Pulses`.

Programmatic Use
Block Parameter:`NumPulses`
Type:double
Values:`positive scalar`
Default:`1`

Data Types: `double`

Enable PRF Output — Enable output of PRF
`off` (default) | `on`

Select this parameter to enable the PRF output port.

Dependencies

To enable this parameter, set **Output signal format** to `Pulses`.

Programmatic Use
Block Parameter:`PRFOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Enable Matched Filter Coeficients Output — Enable output of matched filter coefficients
`off` (default) | `on`

Select this parameter to enable the `Coeff` output port.

Programmatic Use
Block Parameter:`CoefficientOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

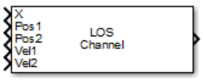
Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also
`phased.LinearFMWaveform`

LOS Channel

Narrowband line-of-sight propagation channel



Libraries:
Phased Array System Toolbox / Environment and Target

Description

The LOS Channel block propagates signals from a point in space to another point in space by line-of-sight (LOS) channels. The block can also propagate signals from one point to multiple points or from multiple points back to one point. The block models propagation time, free-space propagation loss, Doppler shift, and atmospheric as well as weather loss. The block assumes that the propagation speed is much greater than the object speed in which case the stop-and-hop model is valid.

When propagating a signal in an LOS channel to an object and back, you can use a single block to compute two-way LOS channel propagation delays or two blocks to perform two one-way propagation delays in each direction. Because the LOS channel propagation delay is not necessarily an integer multiple of the sampling interval, it can turn out that the total round trip delay in samples when you use a two-way propagation block differs from the delay in samples when you use two one-way propagation blocks. For this reason, it is recommended that, when possible, you use a single two-way propagation block.

Ports

Input

X — Narrowband signal
M-element complex-valued column vector, *M*-by-*N* complex-valued matrix.

Narrowband signal, specified as an *M*-element complex-valued column vector or *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values of the signal and *N* is the number of signals to propagate. When you specify *N* signals, you need to specify *N* signal origins or *N* signal destinations.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Pos1 — Signal origin
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Signal origin, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of propagated signals and equals the dimension specified in the signal input port X. If `Pos1` is a column vector, it takes the form `[x; y; z]`. If `Pos1` is a matrix, each column specifies a different signal origin and has the form `[x; y; z]`. `Pos1` and `Pos2` cannot both be specified as matrices — at least one must be a 3-by-1 column vector. Position units are meters.

Data Types: `double`

Pos2 — Signal destination
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Signal destination, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of propagated signals and equals the dimension specified in the signal in the input port X. If `Pos2` is a column vector, it takes the form `[x; y; z]`. If `Pos2` is a matrix, each column specifies a different signal origin and has the form `[x; y; z]`. `Pos2` and `Pos1` cannot both be specified as matrices — at least one must be a 3-by-1 column vector. Position units are meters.

Data Types: `double`

Vel1 — Signal origin velocity
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Signal origin velocity, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of propagated signals and equals the dimension specified in the signal into the port X. If `Vel1` is a column vector, it takes the form `[Vx; Vy; zV]`. If `Vel1` is a matrix, each column specifies a different signal origin and has the form `[Vx; Vy; Vz]`. The dimensions of `Vel1` must agree with the dimensions of `Pos1`. `Vel1` and `Vel2` cannot both be specified as matrices — at least one must be a 3-by-1 column vector. Position units are meters.

Data Types: `double`

Vel2 — Signal destination velocity
3-by-1 real-valued column vector | 3-by-*N* real-valued matrix

Signal destination velocity, specified as a 3-by-1 real-valued column vector or 3-by-*N* real-valued matrix. The quantity *N* is the number of propagated signals and equals the dimension specified in the signal into the port X. If `Vel2` is a column vector, it takes the form `[Vx; Vy; zV]`. If `Vel2` is a matrix, each column specifies a different signal origin and has the form `[Vx; Vy; Vz]`. The dimensions of `Vel2` must agree with the dimensions of `Pos2`. `Vel2` and `Vel1` cannot both be specified as matrices — at least one must be a 3-by-1 column vector. Position units are meters.

Data Types: `double`

Output

Port_1 — Propagated narrowband signal
M-element complex-valued column vector, *M*-by-*N* complex-valued matrix.

Propagated signal, returned as a *M*-element complex-valued column vector, *M*-by-*N* complex-valued matrix. Y will have the same dimensions as X.

The output Y contains signal samples arriving at the signal destination within the current time frame. The current time frame is defined as the time spanned by the current input. Whenever it takes longer than the current time frame for the signal to propagate from the origin to the destination, the output contains no contribution from the input of the current time frame.

Parameters

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`.

Data Types: `double`

Signal carrier frequency (Hz) — Signal carrier frequency
`3e8` (default) | positive real-valued scalar

Signal carrier frequency, specified as a positive real-valued scalar. Units are in hertz.

Data Types: `double`

Specify atmospheric parameters — Enable atmospheric attenuation model
off (default) | on

Select this parameter to enable to add signal attenuation caused by atmospheric gases, rain, fog, or clouds. When you select this parameter, the **Temperature (degrees Celsius)**, **Dry air pressure (Pa)**, **Water vapour density (g/m^3)**, **Liquid water density (g/m^3)**, and **Rain rate (mm/hr)** parameters appear in the dialog box.

Data Types: `Boolean`

Temperature (degrees Celsius) — Ambient temperature
`15` (default) | real-valued scalar

Ambient temperature, specified as a real-valued scalar. Units are in degrees Celsius.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `double`

Dry air pressure (Pa) — Atmospheric dry air pressure
`101.325e3` (default) | positive real-valued scalar

Atmospheric dry air pressure, specified as a positive real-valued scalar. Units are in pascals (Pa). The default value of this parameter corresponds to one standard atmosphere.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `double`

Water vapour density (g/m^3) — Atmospheric water vapor density
`7.5` (default) | positive real-valued scalar

Atmospheric water vapor density, specified as a positive real-valued scalar. Units are in g/m^3.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `datetime`

Liquid water density (g/m^3) — Liquid water density
`0.0` (default) | nonnegative real-valued scalar

Liquid water density of fog or clouds, specified as a nonnegative real-valued scalar. Units are in g/m^3. Typical values for liquid water density are 0.05 for medium fog and 0.5 for thick fog.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `double`

Rain rate (mm/hr) — Rainfall rate
`0.0` (default) | non-negative real-valued scalar

Rainfall rate, specified as a nonnegative real-valued scalar. Units are in mm/hr.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `double`

Perform two-way propagation — Enable two-way propagation

`off` (default) | `on`

Select this check box to perform round-trip propagation between the origin and destination. Otherwise the block performs one-way propagation from the origin to the destination.

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Maximum one-way propagation distance (m) — Maximum one-way propagation distance

`10e3` (default) | positive real-valued scalar

Specify the maximum distance between the signal origin and the destination as a positive scalar. Units are in meters. Amplitudes of any signals that propagate beyond this distance will be set to zero.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

More About

Attenuation and Loss Factors

Attenuation or path loss in the Wideband LOS channel consists of four components. $L = L_{fsp}L_gL_cL_r$, where

- L_{fsp} is the free-space path attenuation
- L_g is the atmospheric path attenuation
- L_c is the fog and cloud path attenuation

- L_r is the rain path attenuation

Each component is in magnitude units, not in dB.

Propagation Delay, Doppler, and Free-Space Path Loss

When the origin and destination are stationary relative to each other, you can write the output signal of a free-space channel as $Y(t) = x(t-\tau)/L_{fsp}$. The quantity τ is the signal delay and L_{fsp} is the free-space path loss. The delay τ is given by R/c, where R is the propagation distance and c is the propagation speed. The free-space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far field of the transmitting element or array. In the near field, the free-space path loss formula is not valid and can result in a loss smaller than one, equivalent to a signal gain. Therefore, the loss is set to unity for range values, $R \le \lambda/4\pi$.

When the origin and destination have relative motion, the processing also introduces a Doppler frequency shift. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The quantity v is the relative speed of the destination with respect to the origin.

Atmospheric Gas Attenuation Model

This model calculates the attenuation of signals that propagate through atmospheric gases.

Electromagnetic signals attenuate when they propagate through the atmosphere. This effect is due primarily to the absorption resonance lines of oxygen and water vapor, with smaller contributions coming from nitrogen gas. The model also includes a continuous absorption spectrum below 10 GHz. The ITU model *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* is used. The model computes the specific attenuation (attenuation per kilometer) as a function of temperature, pressure, water vapor density, and signal frequency. The atmospheric gas model is valid for frequencies from 1–1000 GHz and applies to polarized and nonpolarized fields.

The formula for specific attenuation at each frequency is

$$\gamma = \gamma_o(f) + \gamma_w(f) = 0.1820 f N''(f)\,.$$

The quantity $N''()$ is the imaginary part of the complex atmospheric refractivity and consists of a spectral line component and a continuous component:

$$N''(f) = \sum_i S_i F_i + N''_D(f)$$

The spectral component consists of a sum of discrete spectrum terms composed of a localized frequency bandwidth function, $F(f)_i$, multiplied by a spectral line strength, S_i. For atmospheric oxygen, each spectral line strength is

$$S_i = a_1 \times 10^{-7}\left(\frac{300}{T}\right)^3 \exp\left[a_2(1 - \left(\frac{300}{T}\right)\right]P\,.$$

For atmospheric water vapor, each spectral line strength is

$$S_i = b_1 \times 10^{-1}\left(\frac{300}{T}\right)^{3.5}\exp\left[b_2(1 - \left(\frac{300}{T}\right)\right]W\,.$$

P is the dry air pressure, W is the water vapor partial pressure, and T is the ambient temperature. Pressure units are in hectoPascals (hPa) and temperature is in degrees Kelvin. The water vapor partial pressure, W, is related to the water vapor density, ρ, by

$$W = \frac{\rho T}{216.7}\,.$$

The total atmospheric pressure is $P + W$.

For each oxygen line, S_i depends on two parameters, a_1 and a_2. Similarly, each water vapor line depends on two parameters, b_1 and b_2. The ITU documentation cited at the end of this section contains tabulations of these parameters as functions of frequency.

The localized frequency bandwidth functions $F_i(f)$ are complicated functions of frequency described in the ITU references cited below. The functions depend on empirical model parameters that are also tabulated in the reference.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length, R. Then, the total attenuation is $L_g= R(\gamma_o + \gamma_w)$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Fog and Cloud Attenuation Model

This model calculates the attenuation of signals that propagate through fog or clouds.

Fog and cloud attenuation are the same atmospheric phenomenon. The ITU model, *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog* is used. The model computes the specific attenuation (attenuation per kilometer), of a signal as a function of liquid water density, signal frequency, and temperature. The model applies to polarized and nonpolarized fields. The formula for specific attenuation at each frequency is

$$\gamma_c = K_l(f)M,$$

where M is the liquid water density in gm/m^3. The quantity $K_l(f)$ is the specific attenuation coefficient and depends on frequency. The cloud and fog attenuation model is valid for frequencies 10–1000 GHz. Units for the specific attenuation coefficient are (dB/km)/(g/m^3).

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length R. Total attenuation is $L_c = R\gamma_c$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply narrowband attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Rainfall Attenuation Model

This model calculates the attenuation of signals that propagate through regions of rainfall. Rain attenuation is a dominant fading mechanism and can vary from location-to-location and from year-to-year.

Electromagnetic signals are attenuated when propagating through a region of rainfall. Rainfall attenuation is computed according to the ITU rainfall model *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. The model computes the specific attenuation (attenuation per kilometer) of a signal as a function of rainfall rate, signal frequency, polarization, and path elevation angle. The specific attenuation, γ_R, is modeled as a power law with respect to rain rate

$$\gamma_R = kR^{\alpha},$$

where R is rain rate. Units are in mm/hr. The parameter k and exponent α depend on the frequency, the polarization state, and the elevation angle of the signal path. The specific attenuation model is valid for frequencies from 1–1000 GHz.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the an effective propagation distance, d_{eff}. Then, the total attenuation is $L = d_{\mathrm{eff}}\gamma_R$.

The effective distance is the geometric distance, d, multiplied by a scale factor

$$r = \frac{1}{0.477d^{0.633}R_{0.01}^{0.073\alpha}f^{0.123} - 10.579(1 - \exp(-0.024d))}$$

where f is the frequency. The article *Recommendation ITU-R P.530-17 (12/2017): Propagation data and prediction methods required for the design of terrestrial line-of-sight systems* presents a complete discussion for computing attenuation.

The rain rate, R, used in these computations is the long-term statistical rain rate, $R_{0.01}$. This is the rain rate that is exceeded 0.01% of the time. The calculation of the statistical rain rate is discussed in *Recommendation ITU-R P.837-7 (06/2017): Characteristics of precipitation for propagation modelling*. This article also explains how to compute the attenuation for other percentages from the 0.01% value.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Version History

Introduced in R2016a

See Also

`phased.LOSChannel`

Matched Filter

Matched filter

Libraries:
Phased Array System Toolbox / Detection

Description

The Matched Filter block implements matched filtering of an input signal. Matched filtering is an FIR filtering operation with the coefficients equal to the time reversed samples of the transmitted signal. The filter can improve SNR before detection.

Ports

Input

X — Input signal
M-by-*N* complex valued matrix

Input signal, specified as an *M*-by-*N* complex valued matrix. Matched filtering is applied along the first dimension. The input X and the output Y have the same dimensions. The initial transient is removed from the filtered result.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Coeff — Matched filter coefficients
M-length complex column vector

Matched filter coefficients, specified as an *M*-length complex column vector.

Dependencies

To enable this port, set the **Source of coefficients** pull-down menu `Input port`.

Data Types: `double`

Output

Y — Matched filter output
M-by-*N* complex valued matrix

Matched filter output, returned as an *M*-by-*N* complex valued matrix. The dimensions of Y are the same as the dimensions of X.

Data Types: `double`

G — Matched-filter gain
N-length real-valued

Matched-filter gain, returned as an *N* real-valued matrix. Gain is computed for each column of X. Units are in dB.

Dependencies

To enable this port, select the **Enable SNR gain output** check box.

Data Types: `double`

Parameters

Source of coefficients — Source of matched filter coefficients

`Property` (default) | `Input port`

Specify whether the matched filter coefficients come from **Coefficients** or from an input port.

`Property`	Matched filter coefficients are specified by **Coefficients** .
`Input port`	Matched filter coefficients are specified via the input port `Coeff` .

Coefficients — Matched filter coefficients

`[1;1]` (default)

Specify the matched filter coefficients as a column vector. This parameter appears when you set **Source of coefficients** to `Property`.

Spectrum window — Window for spectrum weighting

`None` (default) | `Hamming` | `Chebyshev` | `Hann` | `Kaiser` | `Taylor`

Specify the window used for spectrum weighting using one of

`None`
`Hamming`
`Chebyshev`
`Hann`
`Kaiser`
`Taylor`

Spectrum weighting is often used with linear FM waveforms to reduce sidelobe levels in the time domain. The block computes the window length internally to match the FFT length.

Spectrum window range (Hz) — Spectrum window coverage region

`[0, 1e5]` (default)

Specify the spectrum region, in hertz, on which the spectrum window is applied as a 1-by-2 vector in the form of `[StartFrequency,EndFrequency]`.

Note that both `StartFrequency` and `EndFrequency` are measured in baseband. That is, they are within `[-Fs/2,Fs/2]`, where `Fs` is the sample rate specified in any of the waveform library blocks. The parameter `StartFrequency` must be less than `EndFrequency`.

Dependencies

This parameter appears when you set the **Spectrum window** parameter to any value other than `None`.

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.

Programmatic Use
Block Parameter:`SampleRate`
Type:double
Values:positive scalar
Default:`1e6`

Sidelobe attenuation level — Window sidelobe attenuation level

`30` (default)

Specify the sidelobe attenuation level, in dB, of a Chebyshev or Taylor window as a positive scalar.

Dependencies

This parameter appears when you set **Spectrum window** to `Chebyshev` or `Taylor`.

Kaiser shape parameter — Kaiser shape parameter

`0.5` (default)

Specify the parameter that affects the Kaiser window sidelobe attenuation as a nonnegative scalar. Please refer to the function `kaiser` for more details.

Dependencies

This parameter appears when you set the **Spectrum window** parameter to `Kaiser`.

Number of constant level sidelobes — Number of nearly constant sidelobes in Taylor window

4 (default)

Specify the number of nearly-constant-level sidelobes adjacent to the mainlobe in a Taylor window as a positive integer.

Dependencies

This parameter appears when you set the **Spectrum window** parameter to `Taylor`.

Enable SNR gain output — Enable SNR gain output

off (default) | on

Select this check this box to obtain the matched filter SNR gain via the output port `G`. The output port appears only when this box is selected.

Simulate using — Block simulation method

`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use

Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

Introduced in R2014b

See Also

`phased.MatchedFilter`

MFSK Waveform

Multiple frequency shift keying (MFSK) continuous waveform

Libraries:
Phased Array System Toolbox / Waveforms

Description

The MFSK Waveform block generates a multiple frequency-shift keying (MFSK) continuous waveform with a specified step time, sweep bandwidth, frequency offset, and number of steps. The block outputs an integer number of samples, steps, or sweeps. For details on the structure of an MFSK waveform, see `phased.MFSKWaveform`.

Ports

Output

Port_1 — MFSK Waveform
complex-valued column vector

MFSK waveform, returned as a complex-valued column vector containing the waveform samples.

- If the **Output signal format** parameter is set to `Steps`, the output of the port consists of the number of steps set by the **Number of frequency steps in output** parameter.
- If the **Output signal format** parameter is set to `Samples`, the output of the port consists of the number of samples set by the **Number of samples in output** parameter.
- If the **Output signal format** parameter is set to `Sweeps`, the output of the port consists of the number of sweeps set by the **Number of sweeps in output** parameter.

Parameters

Sample rate (Hz) — Sample rate

`151e6` (default) | positive scalar

Sample rate of the signal, specified as a positive scalar. Units are in hertz.

Sweep bandwidth (Hz) — MFSK sweep bandwidth

`150e6` (default) | positive scalar

Bandwidth of the MFSK sweep, specified as a positive scalar. Units are in hertz.

Frequency step burst time (s) — Frequency step burst time

`5e-6` (default) | positive scalar

Time duration of each frequency step, specified as a positive scalar. Units are in seconds.

Number of steps per sweep — Total number of frequency steps

`512` (default) | even positive scalar

Total number of steps in each sweep, specified as an even positive integer.

Chirp offset frequency (Hz) — Chirp offset frequency

`-294e3` (default) | real scalar

Chirp offset frequency, specified as a real scalar. Units are in hertz. The offset determines the frequency translation between the two sequences.

Output signal format — Output signal format

`Steps` (default) | `Samples` | `Sweeps`

Format of the output signal, specified as one of the following:

- `Steps` — The block outputs the number of samples contained in an integer number of frequency steps specified by the **Number of frequency steps in output** parameter.
- `Samples` — The block outputs the number of samples specified in the **Number of samples in output** parameter.
- `Sweeps` — The block outputs the number of samples contained in an integer number of sweeps specified by the **Number of sweeps in output** parameter.

Number of sweeps in output — Number of sweeps in output

`1` (default) | positive integer

Number of sweeps in the block output, specified as a positive integer.

Dependencies

This parameter appears only when you set **Output signal format** to `Sweeps`.

Number of frequency steps in output — Number of frequency steps in output

`1` (default) | positive integer

Number of steps in the block output, specified as a positive integer.

Dependencies

This parameter appears only when you set **Output signal format** to `Steps`.

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use
Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

Introduced in R2015a

See Also

`phased.MFSKWaveform`

Monopulse Feed

Create monopulse sum and difference channels

Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The Monopulse Feed block forms the sum and difference channels used for amplitude monopulse directing finding. Sum and difference channels are derived from signals received by an array. You can feed these channels into the Monopulse Estimator block.

Ports

Input

X — Input signal
complex-valued *M*-by-*N* matrix

Input signal, specified as a complex-valued *M*-by-*N* matrix, where *M* is the number of samples or snapshots of data, and *N* is the number of array elements. If the array contains subarrays, then *N* is the number of subarrays.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

STEER — Array steering direction
scalar | real-valued 2-by-1 column vector

Array steering direction, specified as a scalar or real-valued 2-by-1 column vector.

- When you set the **Monopulse coverage** parameter to `Azimuth`, the steering direction is a scalar and represents the azimuth steering angle.
- When you set the **Monopulse coverage** parameter to `3D`, the steering direction vector has the form `[azimuthAngle; elevationAngle]`, where `azimuthAngle` is the azimuth steering angle, and `elevationAngle` is the elevation steering angle.

Units are in degrees. Azimuth angles lie between –180° and 180°, inclusive, and elevation angles lie between –90° and 90°, inclusive.

Example: `[40;10]`

Data Types: `double`

Output

SIGMA — Sum-channel signal
complex-valued *M*-by-1 column vector

Sum-channel signal, returned as a complex-valued *M*-by-1 column vector, where *M* is the number of rows of X.

Data Types: `double`
Complex Number Support: Yes

DeltaAz — Azimuth-difference channel signal
complex-valued *M*-by-1 column vector

Azimuth-difference channel signal, returned as a complex-valued *M*-by-1 column vector, where *M* is the number of rows of X.

Data Types: `double`
Complex Number Support: Yes

DeltaEl — Elevation-difference channel signal
complex-valued *M*-by-1 vector

Elevation difference-channel signal, returned as a complex-valued *M*-by-1 column vector, where *M* is the number of rows of X.

Dependencies

To enable this output port, set the **Monopulse coverage** parameter to `3D`.

Data Types: `double`
Complex Number Support: Yes

ANG — Estimated direction of target
real-valued 2-by-1 vector

Estimated direction of target, returned as a real-valued 2-by-1 vector in the form `[azimuth,elevation]`. Units are in degrees.

Dependencies

To enable this output port, select the **Output angle estimate** check box.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Monopulse coverage — Monopulse coverage directions
`3D` (default) | `Azimuth`

Coverage directions of monopulse feed, specified as `3D` or `Azimuth`. When you set this parameter to `3D`, the monopulse feed forms the sum channel and both azimuth and elevation difference channels. When you set this parameter to `Azimuth`, the monopulse feed forms the sum channel and the azimuth difference channel.

Squint angle (degrees) — Squint angle
`10` (default) | scalar | real-valued 2-by-1 vector

Squint angle, specified as a scalar or real-valued 2-by-1 vector. The squint angle is the separation angle between the sum beam and the beams along the azimuth and elevation directions.

- When you set the `Monopulse coverage` parameter to `Azimuth`, set the `Squint angle` parameter to a scalar.
- When you set the `Monopulse coverage` parameter to `3D`, you can specify the squint angle as either a scalar or vector. If you set the `Squint angle` parameter to a scalar, the squint angle is the same along both the azimuth and elevation directions. If you set the `Squint angle` parameter to a 2-by-1 vector, its elements specify the squint angle along the azimuth and elevation directions.

Example: `[20;5]`

Output angle estimate — Enable angle estimate output
`off` (default) | `on`

Select this check box to output an estimate of the target direction angle using the `ANG` output port.

Generate Monopulse Tracker — Create Monopulse estimator block
button

Click this button to create a Monopulse Estimate block based on the parameters in this block.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.

- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array

- UCA — Uniform circular array
- Conformal Array — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or UCA.

Element spacing (m) — Spacing between array elements
0.5 for ULA arrays and [0.5,0.5] for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form [SpacingBetweenArrayRows,SpacingBetweenArrayColumns].
- When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or URA.

Array axis — Linear axis direction of ULA
y (default) | x | z

Linear axis direction of ULA, specified as y, x, or z. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to ULA.
- This parameter is also enabled when the block only supports ULA arrays.

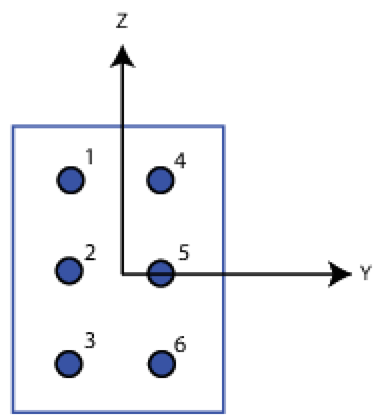
Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting Phase or Time opens the Steer input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting Custom opens the WS input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to Partitioned array or Replicated subarray.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
3.0e8 (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to Partitioned array or Replicated subarray and set **Subarray steering method** to Phase.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
0 (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to Partitioned array or Replicated subarray and set **Subarray steering method** to Phase.

Subarrays layout — Subarray position specification
Rectangular (default) | Custom

Specify the layout of replicated subarrays as Rectangular or Custom.

- When you set this parameter to Rectangular, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to Custom, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

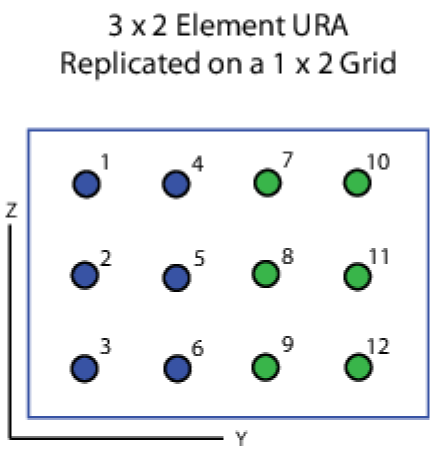
Dependencies

To enable this parameter, set **Sensor array** to Replicated subarray

Grid size — Dimensions of rectangular subarray grid
[1,2] (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

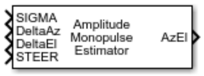
Introduced in R2018b

See Also

`phased.MonopulseFeed`

Monopulse Estimator

Estimate target direction from sum and difference channels



Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The Monopulse Estimator estimates the direction of arrival of a narrowband signal based on an initial guess by applying amplitude monopulse processing on sum and difference channel signals received by an array. You can create these channels using the Monopulse Feed block.

Ports

Input

SIGMA — Sum-channel signal
complex-valued *N*-by-1 column vector

Sum-channel signal, specified as a complex-valued *N*-by-1 column vector. *N* is the number of snapshots in the signal.

Data Types: `double`
Complex Number Support: Yes

DeltaAz — Azimuth difference-channel signal
complex-valued *N*-by-1 column vector

Azimuth difference-channel signal, specified as a complex-valued *N*-by-1 column vector. *N* is the number of snapshots in the signal.

Data Types: `double`
Complex Number Support: Yes

DeltaEl — Elevation difference-channel signal
complex-valued *M*-by-1

Elevation difference-channel signal, specified as a complex-valued *N*-by-1 column vector. *N* is the number of snapshots in the signal.

Dependencies

To enable this output port, set the **Monopulse coverage** parameter to `3D`.

Data Types: `double`
Complex Number Support: Yes

STEER — Array steering direction
scalar | real-valued 2-by-1 column vector

Array steering direction, specified as a scalar or real-valued 2-by-1 column vector.

- When you set the **Monopulse coverage** parameter to `Azimuth`, the steering direction is a scalar and represents the azimuth steering angle.
- When you set the **Monopulse coverage** parameter to `3D`, the steering direction vector has the form `[azimuthAngle; elevationAngle]`, where `azimuthAngle` is the azimuth steering angle, and `elevationAngle` is the elevation steering angle.

Units are in degrees. Azimuth angles lie between –180° and 180°, inclusive, and elevation angles lie between –90° and 90°, inclusive.

Example: `[40;10]`

Data Types: `double`

Output

Az — Estimated azimuth direction of target
real-valued 1-by-*N* vector

Estimated azimuth direction of target, returned as a real-valued 1-by-*N*. The vector elements contain the estimated target direction azimuth angle at each signal snapshot. Units are in degrees.

Dependencies

To enable this output port, set the **Monopulse coverage** to `Azimuth` and the **OutputFormat** to `Angle`.

Data Types: `double`

dAz — Estimated offset of azimuth direction of target
real-valued 1-by-*N* vector

Estimated offset of azimuth direction of target, returned as a real-valued 1-by-*N* vector. The vector elements contain the offset of the estimated target direction azimuth angle from the azimuth steering direction at each signal snapshot. Units are in degrees.

Dependencies

To enable this output port, set the **Monopulse coverage** to `Azimuth` and the **OutputFormat** to `Angle offset`.

Data Types: `double`

AzEl — Estimated direction of target
real-valued 2-by-*N* matrix

Estimated direction of target, returned as a real-valued 2-by-*N* matrix. Each column contains the estimated target direction in the form `[azimuthAngle; elevationAngle]` ,where `azimuthAngle` is the estimated azimuth angle, and `elevationAngle` is estimated elevation angle. Units are in degrees.

Dependencies

To enable this output port, set the **Monopulse coverage** to `3D` and the **OutputFormat** to `Angle`.

Data Types: `double`

dAzEl — Estimated offset of direction of target
real-valued 2-by-*N* matrix

Estimated offset of direction of target, returned as a real-valued 2-by-*N* matrix. The offset is the difference between the target direction and the steering vector. Each column contains the estimated offset of the target direction in the form `[dazimuthAngle; delevationAngle]`, where `dazimuthAngle` is the estimated azimuth angle offset, and `delevationAngle` is estimated elevation angle offset. Units are in degrees.

Dependencies

To enable this output port, set the **Monopulse coverage** to `3D` and the **OutputFormat** to `Angle offset`.

Data Types: `double`

AzRatio — Ratio of sum and azimuth difference channels
real-valued 1-by-*N* vector

Ratio of sum and azimuth difference channels, returned as a real-valued 1-by-*N* vector. The elements contain the ratio of the sum to azimuth difference channel at each signal snapshot.

Dependencies

To enable this output port, set the **Monopulse coverage** to `Azimuth` and select the **Output sum difference ratio** check box.

Data Types: `double`

AzElRatio — Ratio of sum channel to azimuth and elevation difference channels
real-valued 2-by-*N* matrix

Ratio of sum and azimuth and elevation difference channels, returned as a real-valued 2-by-*N* matrix. The elements of the first row contain the ratio of the sum to azimuth difference channel at each signal snapshot. The elements of the second row contain the ratio of the sum to elevation difference channel at each signal snapshot.

Dependencies

To enable this output port, set the **Monopulse coverage** to `3D` and select the **Output sum difference ratio** check box.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Monopulse coverage — Monopulse coverage directions
3D (default) | Azimuth

Monopulse coverage directions, specified as 3D or Azimuth. When you set this parameter to 3D, the monopulse estimator uses the sum channel and both azimuth and elevation difference channels. When you set this parameter to Azimuth, the monopulse estimator uses the sum channel and the azimuth difference channel.

Squint angle (degrees) — Squint angle
10 (default) | scalar | real-valued 2-by-1 vector

Squint angle, specified as a scalar or real-valued 2-by-1 vector. The squint angle is the separation angle between the sum beam and the beams along the azimuth and elevation directions.

- When you set the Monopulse coverage parameter to Azimuth, set the Squint angle parameter to a scalar.
- When you set the Monopulse coverage parameter to 3D, you can specify the squint angle as either a scalar or vector. If you set the Squint angle parameter to a scalar, the squint angle is the same along both the azimuth and elevation directions. If you set the Squint angle parameter to a 2-by-1 vector, its elements specify the squint angle along the azimuth and elevation directions.

Example: [20;5]

Output format — Output direction format
Angle (default) | Angle offset

Format of direction output, specified Angle or Angle offset. When you set this parameter to Angle, the output port is labeled AzEl or Az and is the actual direction of the target. When you set this property to Angle offset, the output port is labeled dAzEl or dAz and is the angle offset of the target from the array steering direction.

Output sum difference ratio — Enable sum-difference ratio output port
off (default) | on

Select this check box to output the ratio of the sum and difference channels in the azimuth and elevation directions. When you set the **Monopulse coverage** to Azimuth, the block outputs the sum-azimuth difference ratio using the AzRatio port. When you set the **Monopulse coverage** to 3D, the block outputs the sum-azimuth difference and sum-elevation difference channels ratio using the AzElRatio port.

Generate Monopulse Feed — Create monopulse feed block
button

Click this button to create a Monopulse Feed block based on the parameters in this block.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
ULA (default) | URA | UCA | `Conformal Array`

Array geometry, specified as one of

- ULA — Uniform linear array
- URA — Uniform rectangular array
- UCA — Uniform circular array
- Conformal Array — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or UCA.

Element spacing (m) — Spacing between array elements
0.5 for ULA arrays and [0.5,0.5] for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form [SpacingBetweenArrayRows,SpacingBetweenArrayColumns].
- When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or URA.

Array axis — Linear axis direction of ULA
y (default) | x | z

Linear axis direction of ULA, specified as y, x, or z. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

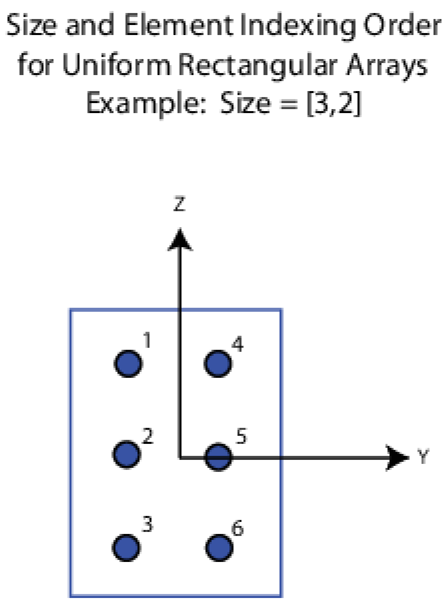
- To enable this parameter, set **Geometry** to ULA.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

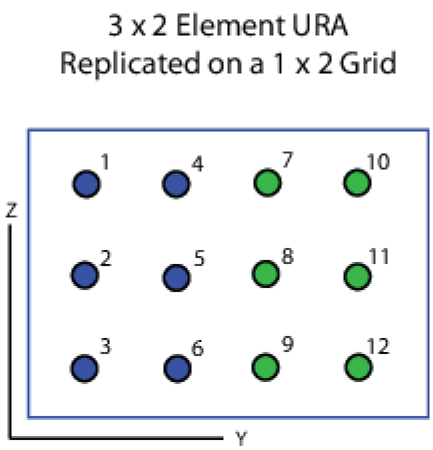
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

Introduced in R2018b

See Also

`phased.MonopulseEstimator`

MVDR Beamformer

Narrowband MVDR (Capon) beamformer

Libraries:
Phased Array System Toolbox / Beamforming

Description

The MVDR Beamformer block performs minimum variance distortionless response (MVDR) beamforming. The block preserves the signal power in the given direction while suppressing interference and noise from other directions. The MVDR beamformer is also called the Capon beamformer.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

XT — Training signal
M-by-*N* complex-valued matrix

Training signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this port, select the **Enable training data input** check box.

Data Types: `double`

Ang — Beamforming direction
2-by-1 real-valued vector | 2-by-*L* real-valued matrix

Beamforming direction, specified as a 2-by-*L* real-valued matrix, where *L* is the number of beamforming directions. Each column takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Source of beamforming direction** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-*L* complex-valued matrix

Beamformed output, returned as an *M*-by-*L* complex-valued matrix. The quantity *M* is the number of signal samples and *L* is the number of desired beamforming directions specified by the `Beamforming direction` parameter or from the `Ang` port.

Data Types: `double`

W — Beamforming weights
N-by-*L* complex-valued matrix

Beamformed weights, returned as an *N*-by-*L* complex-valued matrix. The quantity *N* is the number of array elements. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays. *L* is the number of desired beamforming directions specified in the `Ang` port or by the `Beamforming direction (deg)` property. There is one set of weights for each beamforming direction.

Dependencies

To enable this port, select the **Enable weights output** checkbox.

Data Types: `double`

Parameters

Main tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: 3e8

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Diagonal loading factor — Diagonal loading factor for stability
nonnegative scalar

Specify the diagonal loading factor as a nonnegative scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample support is small.

Enable training data input — Enable the use of training data
off (default) | on

Select this check box to specify additional training data via the input port `XT`. To use the input signal as the training data, clear the check box which removes the port.

Source of beamforming direction — Source of beamforming direction
`Property` (default) | `Input port`

Source of beamforming direction, specified as `Property` or `Input port`. When you set **Source of beamforming direction** to `Property`, you then set the direction using the **Beamforming direction (deg)** parameter. When you select `Input port`, the direction is determined by the input to the `Ang` port.

Beamforming direction (deg) — Beamforming directions
2-by-*L* real-valued matrix

Beamforming directions, specified as a *2*-by-*L* real-valued matrix, where *L* is the number of beamforming directions. Each column takes the form `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this parameter, set the **Source of beamforming direction** parameter to `Property`.

Number of bits in phase shifters — Number of phase shift quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><th rowspan="2">Block Simulation</th><th colspan="3">Simulation Behavior</th></tr>
<tr><th><code>Normal</code></th><th><code>Accelerator</code></th><th><code>Rapid Accelerator</code></th></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-L row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.

- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
ULA (default) | URA | UCA | `Conformal Array`

Array geometry, specified as one of

- ULA — Uniform linear array
- URA — Uniform rectangular array

- UCA — Uniform circular array
- Conformal Array — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or UCA.

Element spacing (m) — Spacing between array elements
0.5 for ULA arrays and [0.5,0.5] for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form [SpacingBetweenArrayRows,SpacingBetweenArrayColumns].
- When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or URA.

Array axis — Linear axis direction of ULA
y (default) | x | z

Linear axis direction of ULA, specified as y, x, or z. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

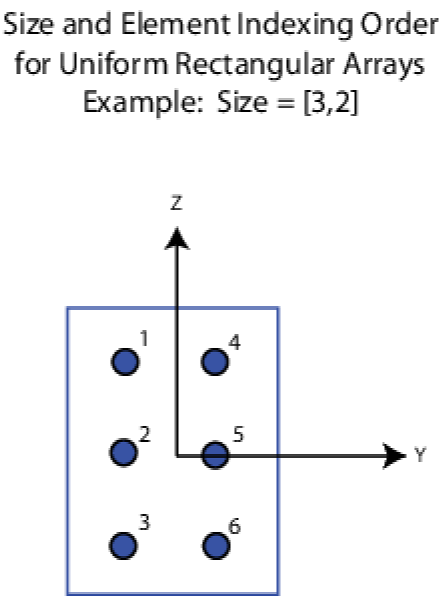
- To enable this parameter, set **Geometry** to ULA.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting Phase or Time opens the Steer input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting Custom opens the WS input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to Partitioned array or Replicated subarray.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
3.0e8 (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to Partitioned array or Replicated subarray and set **Subarray steering method** to Phase.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
0 (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to Partitioned array or Replicated subarray and set **Subarray steering method** to Phase.

Subarrays layout — Subarray position specification
Rectangular (default) | Custom

Specify the layout of replicated subarrays as Rectangular or Custom.

- When you set this parameter to Rectangular, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to Custom, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

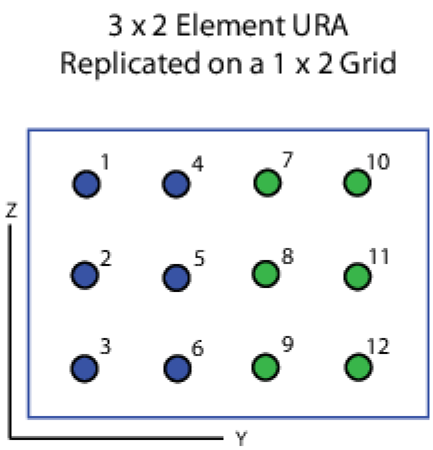
Dependencies

To enable this parameter, set **Sensor array** to Replicated subarray

Grid size — Dimensions of rectangular subarray grid
[1,2] (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

Introduced in R2014b

See Also

`phased.MVDRBeamformer`

MUSIC Spectrum

MUSIC 2D spatial spectrum estimator

Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The MUSIC Spectrum block uses the MUltiple SIgnal Classification (MUSIC) algorithm to estimate the spatial spectrum of incoming narrowband signals. The block optionally calculates the direction of arrival of a specified number of signals by finding the peaks of the spectrum.

Ports

Input

Port 1 — Received signal
M-by-*N* complex-valued matrix

Received signal, specified as an *M*-by-*N* complex-valued matrix. The quantity *M* is the length of the signal, the number of sample values contained in the signal. The quantity *N* is the number of sensor elements in the array.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Output

Y — MUSIC 2-D spatial spectrum
non-negative real-valued *P*-by-*Q* matrix

2-D MUSIC spatial spectrum, returned as a non-negative, returned as a real-valued *P*-by-*Q* matrix. Each entry represents the magnitude of the estimated MUSIC spatial spectrum. Each entry corresponds to an angle specified by the **Azimuth scan angles (deg)** and **Elevation scan angles (deg)** parameters. *P* equals the length of the vector specified in **Azimuth scan angles (deg)** and *Q* equals the length of the vector specified in **Elevation scan angles (deg)**.

Data Types: `double`

Ang — Directions of arrival
non-negative, real-valued 2-by-*L* matrix

Directions of arrival of the signals, returned as a real-valued 2-by-*L* matrix. *L* is the number of signals specified by the **Number of signals** parameter. The direction of arrival angle is defined by the azimuth and elevation angles of the source with respect to the array local coordinate system. The first row of the matrix contains the azimuth angles and the second row contains the elevation angles. If the object cannot identify peaks in the spectrum, it will return `NaN`. Angle units are in degrees.

Dependencies

Select the **Enable DOA output** parameter to enable this output port.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Forward-backward averaging — Enable forward-backward averaging
off (default) | on

Select this parameter to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold structure.

Azimuth scan angles (deg) — Azimuth scan angles
`-90:90` (default) | real-valued scalar | real-valued row vector

Azimuth scan angles, specified as a real-valued row vector. The angle values must lie between –180° and 180°, inclusive, and specified in ascending order. Angle units are in degrees.

Elevation scan angles (deg) — Elevation scan angles
`0` (default) | real-valued scalar | real-valued row vector

Elevation scan angles, specified as a scalar or real-valued row vector. The angle values must lie between –90° and 90°, inclusive, and specified in ascending order. Angle units are in degrees.

Enable DOA output — Output directions of arrival through output port
off (default) | on

Select this parameter to output the signals directions of arrival (DOA) through the **Ang** output port.

Number of signals — Expected number of arriving signals
`1` (default) | positive integer

Specify the expected number of signals for DOA estimation as a positive scalar integer.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><td rowspan="2">Block Simulation</td><td colspan="3">Simulation Behavior</td></tr>
<tr><td><code>Normal</code></td><td><code>Accelerator</code></td><td><code>Rapid Accelerator</code></td></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
ULA (default) | URA | UCA | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

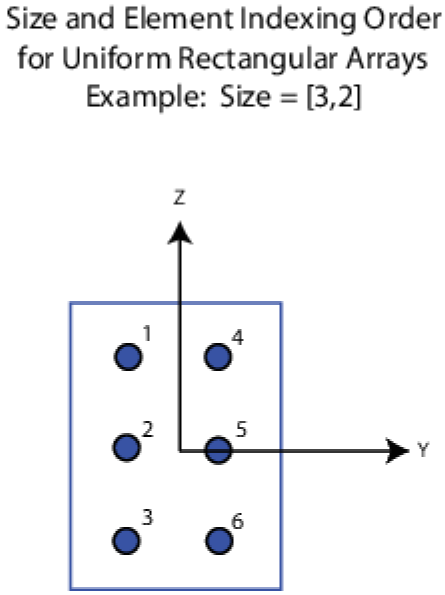
Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.

- If **Array size** is an integer, the array has the same number of elements in each row and column.

For a URA, array elements are indexed from top to bottom along the leftmost array column, and continued to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to URA.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to URA.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Data Types: `double`

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. If the parameter value is a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

To enable this parameter, set **Geometry** to `Conformal Array`.

Data Types: `double`

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Version History
Introduced in R2016b

See Also

Blocks
ULA MUSIC Spectrum

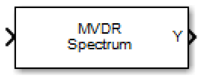
Objects
`phased.MUSICEstimator2D` | `phased.MUSICEstimator` | `phased.ConformalArray` | `phased.UCA` | `phased.ULA` | `phased.URA`

Functions
`musicdoa`

Topics
"MUSIC Super-Resolution DOA Estimation"

MVDR Spectrum

Minimum variation distortionless response (MVDR) spatial spectrum estimator



Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The narrowband MVDR Spectrum block estimates the spatial spectrum of incoming narrowband signals by scanning a range of azimuth and elevation angles using an MVDR conventional beamformer. The block optionally calculates the direction of arrival of a specified number of signals by estimating the peaks of the spectrum. This estimator is also referred to as a Capon estimator.

Ports

Input

In — Input data
complex-valued matrix

Input data, specified as a complex-valued matrix whose columns correspond to channels. Rows correspond to samples.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double` | `single`

Output

Y — Two-dimensional spatial spectrum
real-valued matrix

Spatial spectrum, returned as a real-valued matrix representing the magnitude of the estimated 2-D spatial spectrum. The row dimension of Y is equal to the number of angles in the **Elevation scan angles (deg)** parameter and the column dimension of Y is equal to the number of angles in the **Azimuth scan angles (deg)** parameter.

Data Types: `double`

Ang — Estimated direction of arrival
two-row real-valued matrix

Estimated direction of arrival, returned as a two-row real-valued matrix where the first row represents estimated azimuth angles and the second row represents estimated elevation angles. Units are in degrees.

Dependencies

To enable this port, select the **Enable DOA output** check box.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Number of bits in phase shifters — Number of phase shift quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Forward-backward averaging — Enable forward-backward averaging
off (default) | on

Select this parameter to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold structure.

Azimuth scan angles (deg) — Azimuth scan angles

`-90:90` (default)

Specify the azimuth scan angles, in degrees, as a real vector. The angles must be between –180° and 180°, inclusive. You must specify the angles in ascending order.

Elevation scan angles (deg) — Elevation scan angles

`0` (default)

Specify the elevation scan angles, in degrees, as a real vector or scalar. The angles must be between –90° and 90°, inclusive. You must specify the angles in an ascending order.

Enable DOA output — Output directions of arrival through output port
off (default) | on

Select this parameter to output the signals directions of arrival (DOA) through the **Ang** output port.

Number of signals — Expected number of arriving signals
1 (default) | positive integer

Specify the expected number of signals for DOA estimation as a positive scalar integer.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: phased.URA('Size',[5,3])

Dependencies

To enable this parameter, set **Specify sensor array as** to MATLAB expression.

Sensor Array Tab

Specify sensor array as — Method to specify array
Array (no subarrays) (default) | MATLAB expression

Method to specify array, specified as Array (no subarrays) or MATLAB expression.

- Array (no subarrays) — use the block parameters to specify the array.
- MATLAB expression — create the array using a MATLAB expression.

Element Parameters

Element type — Array element types
Isotropic Antenna (default) | Cardioid Antenna | Cosine Antenna | Custom Antenna | Gaussian Antenna | Sinc Antenna | Omni Microphone | Custom Microphone

Antenna or microphone type, specified as one of the following:

- Isotropic Antenna
- Cardioid Antenna
- Cosine Antenna
- Custom Antenna
- Gaussian Antenna
- Sinc Antenna
- Omni Microphone
- Custom Microphone

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
[0,1e20] (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form [LowerBound,UpperBound]. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to Isotropic Antenna, Cosine Antenna, or Omni Microphone.

Baffle the back of the element — Set back response of an Isotropic Antenna element or an Omni Microphone element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Null axis direction — Null axis direction
`-x` (default) | `+x` | `+y` | `-y` | `+z` | `-z`

Dependencies

To enable this parameter, set **Element type** to `Cardioid Antenna`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-L row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Align element normal with array normal — Align element normal with array normal
on (default) | off

Dependencies

This parameter is enabled when **Element type** is set to `Custom Antenna`.

Radiation pattern beamwidth (deg) — Radiation pattern beamwidth
`[10,10]` (default)

Dependencies

This parameter is enabled when **Element type** is set to `Gaussian Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.

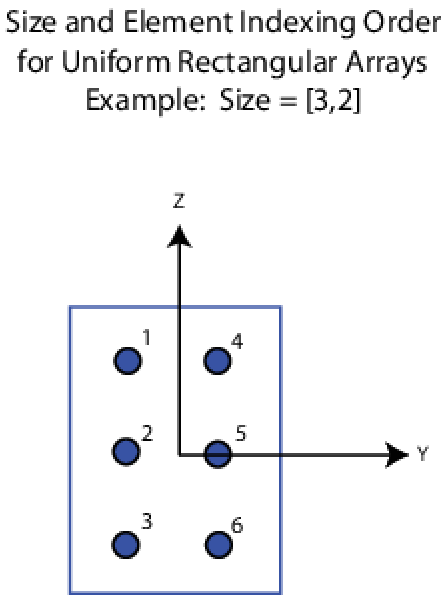
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form [NumberOfArrayRows,NumberOfArrayColumns].
- If **Array size** is an integer, the array has the same number of elements in each row and column.

For a URA, array elements are indexed from top to bottom along the leftmost array column, and continued to the next columns from left to right. In this figure, the **Array size** value of [3,2] creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to URA.

Element lattice — Lattice of URA element positions
Rectangular (default) | Triangular

Lattice of URA element positions, specified as Rectangular or Triangular.

- Rectangular — Aligns all the elements in row and column directions.
- Triangular — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to URA.

Array normal — Array normal direction
x for URA arrays or z for UCA arrays (default) | y

Array normal direction, specified as x, y, or z.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
[0;0;0] (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position [x;y;z]of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Data Types: `double`

Element normals (deg) — Direction of conformal array element normal vectors
[0;0] | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. If the parameter value is a matrix, each column specifies the normal direction of the corresponding element in the form [azimuth;elevation] with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

To enable this parameter, set **Geometry** to `Conformal Array`.

Data Types: `double`

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Version History

Introduced in R2014b

See Also

`phased.MVDREstimator2D`

Narrowband Receive Array

Receive narrowband radiation using phased array

Libraries:
Phased Array System Toolbox / Transmitters and Receivers

Description

The Narrowband Receive Array block implements a narrowband receive array collecting incoming radiation. The array processes narrowband plane waves incident on the sensor elements of the array. The delay at each element is approximated using the corresponding phase shift in the time domain.

Ports

Input

X — Incident signals
complex-valued M-by-Lmatrix

Incident signals, specified as a complex-valued M-by-L matrix, where M is the number of samples in the data, and L is the number of incident signals. Each column of X is a far field signal.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double` | `single`

Ang — Incident signal directions
2-by-1 real-valued vector | 2-by-L real-valued matrix

Incident directions of signals, specified as a real-valued 2-by-L matrix. Each column specifies the incident direction of the corresponding column of X and takes the form [azimuth; elevation]. Units are in degrees. The azimuth angle must lie in the range from –180° to 180°, inclusive. The elevation angle must lie in the range from –90° to 90°, inclusive.

Data Types: `single` | `double`

W — Element or subarray weights
complex-valued P-by-1 vector

Element or subarray weights, specified as a complex-valued P-by-1 column vector where P is the number of array elements (or subarrays when the array supports subarrays).

Dependencies

To enable this port, select the **Enable weights input** check box.

Data Types: `single` | `double`

WS — Subarray element weights
complex-valued N_{SE}-by-L matrix

Subarray element weights, specified as a complex-valued N_{SE}-by-L matrix. N_{SE} is the number of subarrays. L is the number of incident signals. The same weight is applied to the individual elements within a subarray.

Subarray Element Weights

Specify sensor array as:	Subarray Weights
`Replicated subarray`	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of `WS` specifies the weights for the corresponding subarray.
`Partitioned array`	Subarrays can have different dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. If all the subarrays have the same size, $Q = N_{SE}$. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this port, set the **Specify sensor array as** parameter to `Partitioned array` or `Replicated subarray` and set the **Subarray steering method** to `Custom`.

Data Types: `double` | `single`

Steer — Subarray steering angle
real-valued length-2 column vector

Subarray steering angle, specified as a real-valued length-2 column vector. The vector has the form [azimuth; elevation], in degrees. Units are in degrees. The azimuth angle must lie in the range from –180° to 180°, inclusive. The elevation angle must lie in the range from –90° to 90°, inclusive.

Dependencies

To enable this port, set the **Specify sensor array as** parameter to `Partitioned array` or `Replicated subarray` and set the **Subarray steering method** to `Phase` or `Time`.

Data Types: `single` | `double`

Output

Y — Collected signals
complex-valued *M*-by-*P* matrix

Collected signals, returned as a complex-valued *M*-by-*P* matrix. *M* is the length of the input signal. *P* is the number of array elements (or subarrays when subarrays are supported). Each column corresponds to the signal collected by the corresponding array element (or corresponding subarrays when subarrays are supported).

Data Types: `double` | `single`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Sensor gain measure — Sensor gain measure
`dB` (default) | `dBi`

Sensor gain measure, specified as `dB` or `dBi`.

- When you set this parameter to `dB`, the input signal power is scaled by the sensor power pattern (in dB) in the corresponding direction and then combined.
- When you set this parameter to `dBi`, the input signal power is scaled by the directivity pattern (in dBi) in the corresponding direction and then combined. This option is useful when you want to compare results with the values computed by the radar equation that uses dBi to specify the antenna gain. The computation using the `dBi` option is costly as it requires an integration over all directions to compute the total radiated power of the sensor.

Data Types: `char` | `string`

Enable weights input — Option to input weights
`off` (default) | `on`

Select this check box to specify array weights via the input port `W`. The input port appears only when this box is selected.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Clicking the **Analyze** button opens the **Sensor Array Analyzer** app. The app lets you examine important array properties such as array response and array geometry.

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.

- `MATLAB expression` — create the array using a MATLAB expression.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

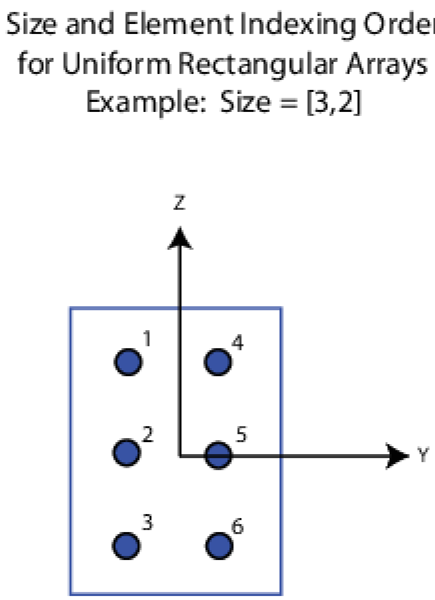
- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

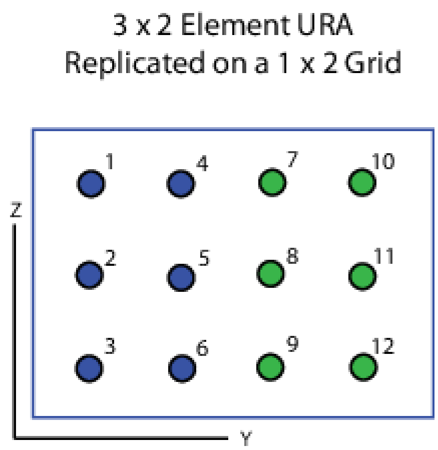
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as ULA or URA.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

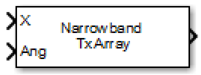
Introduced in R2014a

See Also

`phased.Collector`

Narrowband Transmit Array

Narrowband transmit array



Libraries:
Phased Array System Toolbox / Transmitters and Receivers

Description

The Narrowband Transmit Array block generates narrowband plane waves in the far field of the array by adding the far-field radiated signals of each element. Think of the block output as the field at a reference distance from the element or from the center of the array.

Ports

Input

X — Radiated signals
complex-valued *M*-by-1 vector | complex-valued *M*-by-*N* matrix

Radiate signals, specified as a complex-valued *M*-by-1 vector or complex-valued *M*-by-*N* matrix. *M* is the length of the signal, and *N* is the number of array elements (or subarrays when subarrays are supported).

Dimensions of X

Dimension	Signal
M-by-1 vector	The same signal is radiated from all array elements (or all subarrays when subarrays are supported).
`M-by-N matrix`	Each column corresponds to the signal radiated by the corresponding array element (or corresponding subarrays when subarrays are supported).

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`
Complex Number Support: Yes

Ang — Input
real-valued 2-by-*L* matrix

Radiating directions of signals, specified as a real-valued 2-by-*L* matrix. Each column specifies a radiating direction in the form `[AzimuthAngle;ElevationAngle]`. The azimuth angle must lie between –180° and 180°, inclusive. The elevation angle must lie between –90° and 90°, inclusive. Units are in degrees.

Example: [30,20;45,0]

Data Types: double

W — Element or subarray weights
N-by-1 column vector

Element or subarray weights, specified as a complex-valued N-by-1 column vector where N is the number of array elements (or subarrays when the array supports subarrays).

Dependencies

To enable this port, select the **Enable weights input** check box.

Data Types: double

WS — Subarray element weights input port
complex-valued N_{SE}-by-N matrix | 1-by-N cell array

Subarray element weights, specified as complex-valued N_{SE}-by-N matrix or 1-by-N cell array where N is the number of subarrays. These weights are applied to the individual elements within a subarray.

Subarray element weights

Sensor Array	Subarray weights
Replicated subarray	All subarrays have the same dimensions and sizes. Then, the subarray weights form an N_{SE}-by-N matrix. N_{SE} is the number of elements in each subarray and N is the number of subarrays. Each column of WS specifies the weights for the corresponding subarray.
Partitioned array	Subarrays may not have the same dimensions and sizes. In this case, you can specify subarray weights as • an N_{SE}-by-N matrix, where N_{SE} is now the number of elements in the largest subarray. The first Q entries in each column are the element weights for the subarray where Q is the number of elements in the subarray. • a 1-by-N cell array. Each cell contains a column vector of weights for the corresponding subarray. The column vectors have lengths equal to the number of elements in the corresponding subarray.

Dependencies

To enable this port, specify the `Specify sensor array as` parameter as **Replicated subarray** or **Partitioned array** and then select `Custom` from the **Subarray steering method** drop down menu.

Data Types: double

Steer — Input
real-valued 2-by-1 vector

Subarray steering angle, specified as a length-2 column vector. The vector has the form `[azimuthAngle;elevationAngle]`. The azimuth angle must be between –180° and 180°, inclusive. The elevation angle must be between –90° and 90°, inclusive. Units are in degrees.

Dependencies

To enable this port, select `Phase` or `Time` from the **Subarray steering method** pull down menu.

Data Types: `double`

Output

Output 1 — Output
complex-valued *M*-by-*L* matrix

Radiated signals, specified as a complex-valued *M*-by-*L* matrix, where *L* is the number of radiating angles, `ANG` and *M* is the length of the input signal, `X`.

Data Types: `double`

Parameters

Main Tab

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`.

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Sensor gain measure — Specify sensor gain

`dB` (default) | `dBi`

Specify sensor gain measure as `dB` or `dBi`.

- When you set this parameter to `dB`, the input signal power is scaled by the sensor power pattern (in dB) at the corresponding direction and then combined.
- When you set this parameter to `dBi`, the input signal power is scaled by the directivity pattern (in dBi) at the corresponding direction and then combined. This option is useful when you want to compare results with the values computed by the radar equation that uses dBi to specify the antenna gain. The computation using the `dBi` option is expensive as it requires an integration over all directions to compute the total radiated power of the sensor. The default value is `dB`.

Enable weights input — Enable weights input

off (default) | on

Select this check box to specify array weights using the input port `W`. The input port appears only when this box is checked.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use

Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Single element` | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Specify sensor element or sensor array. A sensor array can also contain subarrays or be a partitioned array. This parameter can also be expressed as a MATLAB expression.

Types

`Single element`
`Array (no subarrays)`
`Partitioned array`
`Replicated subarray`
`MATLAB expression`

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cardioid Antenna` | `Cosine Antenna` | `Custom Antenna` | `Gaussian Antenna` | `Sinc Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cardioid Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Gaussian Antenna`
- `Sinc Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Null axis direction — Null axis direction
-x (default) | +x | +y | -y | +z | -z

Dependencies

To enable this parameter, set **Element type** to `Cardioid Antenna`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Align element normal with array normal — Align element normal with array normal
on (default) | off

Dependencies

This parameter is enabled when **Element type** is set to `Custom Antenna`.

Radiation pattern beamwidth (deg) — Radiation pattern beamwidth
`[10,10]` (default)

Dependencies

This parameter is enabled when **Element type** is set to `Gaussian Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
ULA (default) | URA | UCA | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

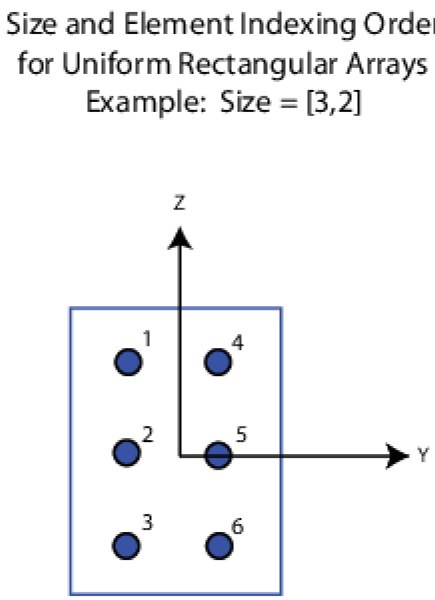
To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to URA.

Element spacing (m) — Spacing between array elements
0.5 for ULA arrays and [0.5,0.5] for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form [SpacingBetweenArrayRows,SpacingBetweenArrayColumns].
- When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or URA.

Array axis — Linear axis direction of ULA
y (default) | x | z

Linear axis direction of ULA, specified as y, x, or z. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to ULA.
- This parameter is also enabled when the block only supports ULA arrays.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
`x`	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
`y`	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
`z`	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

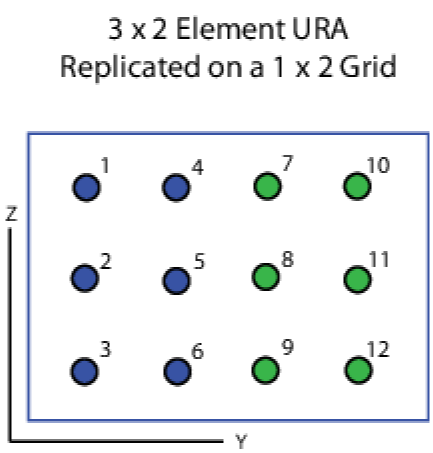
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Analyze — Launch Sensor Array Analyzer
(default)

Clicking the **Analyze** button launches the **Sensor Array Analyzer** app. The app lets you examine important array properties such as array response and array geometry.

Version History

Introduced in R2014b

See Also

`phased.Radiator`

Nonlinear FM Waveform

Nonlinear FM pulse waveform

Libraries:
Phased Array System Toolbox / Waveforms

Description

The Nonlinear FM Waveform block generates a nonlinear FM pulse waveform with specified pulse width, pulse repetition frequency (PRF), and sweep bandwidth. The block outputs an integer number of pulses or an integral number of samples.

Ports

Input

PRFIdx — PRF Index
positive integer

Index to select the pulse repetition frequency (PRF), specified as a positive integer. The index selects the PRF from the predefined vector of values specified by the **Pulse repetition frequency (Hz)** parameter.

Example: 4

Dependencies

To enable this port, select **Enable PRF selection input**.

Data Types: `double`

FreqOffset — Frequency offset
scalar

Frequency offset in Hz, specified as a scalar.

Example: `2e3`

Dependencies

To enable this port, set **Source of Frequency Offset** to `Input port`.

Data Types: `double`

Output

Y — Pulse waveform
complex-valued vector

Pulse waveform samples, returned as a complex-valued vector.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, returned as a positive scalar.

Dependencies

To enable this port, set the **Output signal format** parameter to `Pulses` and then select the **Enable PRF output** parameter.

Data Types: `double`

Coeff — Matched filter coefficients
vector | matrix

Matched filter coefficients, returned as a vector or matrix.

Dependencies

To enable this port, select **Enable Matched Filter Coefficients Output**.

Data Types: `double`

Parameters

To edit block parameters interactively, use the **Property Inspector**. From the Simulink Toolstrip, on the **Simulation** tab, in the **Prepare** gallery, select **Property Inspector**.

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.

Programmatic Use
Block Parameter:`SampleRate`
Type:double
Values:positive scalar
Default:`1e6`

Method to specify pulse duration — Pulse duration as time or duty cycle
`Pulse width` (default) | `Duty cycle`

Method to set the pulse duration, specified as `Pulse width` or `Duty cycle`. When you set this parameter to `Pulse width`, the pulse duration is set using the **Pulse width (s)** parameter. When you set this parameter to `Duty cycle`, the pulse duration is computed from the values of the **Pulse repetition frequency (Hz)** and **Duty Cycle** parameters.

Programmatic Use
Block Parameter:`DurationSpecification`
Type:string
Values:string
Default:`'Pulse width'`

Pulse width (s) — Time duration of pulse
`50e-6` (default) | positive scalar

The duration of each pulse, specified as a positive scalar. Set the product of **Pulse width (s)** and **Pulse repetition frequency** to be less than or equal to one. This restriction ensures that the pulse width is smaller than the pulse repetition interval. Units are in seconds.

Example: `300e-6`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Pulse width`.

Programmatic Use

Block Parameter:`PulseWidth`
Type:double
Values:string
Default:`50e-6`

Duty cycle — Waveform duty cycle
`0.5` (default) | scalar in the range *[0,1]*

Waveform duty cycle, specified as a scalar in the range *[0,1]*.

Example: `0.7`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Duty cycle`.

Programmatic Use

Block Parameter:`DutyCycle`
Type:double
Values:positive scalar
Default:`1e6`

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1e4` (default) | positive scalar

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The value of **Pulse repetition frequency (Hz)** must satisfy these constraints:

- The product of **Pulse width** and **Pulse repetition frequency (Hz)** must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of **Pulse repetition frequency** must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* by using block parameter settings alone or in conjunction with the input port, `PRFIdx`.

- When the **Enable PRF selection input** parameter is not selected, set the *PRF* using block parameters.
  - To implement a constant *PRF*, specify **Pulse repetition frequency (Hz)** as a positive scalar.
  - To implement a staggered *PRF*, specify **Pulse repetition frequency (Hz)** as a row vector with positive values. After the waveform reaches the last element of the vector, the process continues cyclically with the first element of the vector. When *PRF* is staggered, the time between successive output pulses cycles through the successive values of the *PRF* vector.
- When the **Enable PRF selection input** parameter is selected, you can implement a selectable *PRF* by specifying **Pulse repetition frequency (Hz)** as a row vector with positive real-valued entries. But this time, when you execute the block, select a *PRF* by passing an index into the *PRF* vector into the `PRFIdx` port.

In all cases, the number of output samples is fixed when you set the **Output signal format** to `Samples`. When you use a varying *PRF* and set **Output signal format** to `Pulses`, the number of output samples can vary.

Programmatic Use
Block Parameter:PRF
Type:double
Values:positive scalar
Default:`1e6`

Enable PRF selection input — Select predefined PRF
off (default) | on

Select this parameter to enable the `PRFIdx` port.

- When enabled, pass in an index into a vector of predefined PRFs. Set predefined PRFs using the **Pulse repetition frequency (Hz)** parameter.
- When not enabled, the block cycles through the vector of PRFs specified by the **Pulse repetition frequency (Hz)** parameter. If **Pulse repetition frequency (Hz)** is a scalar, the PRF is constant.

Programmatic Use
Block Parameter:`PRFSelectionInputPort`
Type:logical
Values:positive scalar
Default:`off`

Frequency modulation — Frequency modulation
`Polynomial` (default) | `Hyperbolic` | `Hybrid Linear-Tangent` | `Stepped Price`

Frequency modulation of the nonlinear FM waveform, specified as `Polynomial`, `Hyperbolic`, `Hybrid Linear-Tangent`, or `Stepped Price`.

- When set to `Polynomial` the block generates a waveform with an instantaneous frequency that follows a polynomial function. The coefficients of the polynomial are specified by the **Coefficients of a Polynomial FM** parameter. The resulting frequency function is normalized such that each pulse sweeps the bandwidth specified in **Sweep bandwidth (Hz)**. The **Sweep interval** parameter is inactive when **Frequency modulation** is set to `Polynomial`.

- When set to `Hyperbolic` the block generates a hyperbolic frequency modulated (HFM) waveform. Use the **Start frequency of a hyperbolic FM** parameter to set the start frequency of the hyperbolic sweep. The **Sweep interval** parameter is inactive in this case.
- When set to `Hybrid Linear-Tangent` the block generates a hybrid NLFM waveform that combines an LFM with a tan-FM as described by Collins and Atkins [1]. The balance between LFM and tan-FM is specified by the **Balance between linear FM and tan-FM** parameter, and the portion of the *tan(x)* curve used for tan-FM is specified by the **Tangent curve portion** parameter.
- When set to `Stepped Price` the block generates a stepped version of the Price's NLFM waveform as given by Levanon and Mozeson [2]. The **Sweep interval** parameter is inactive when **Frequency modulation** parameter is set to `Stepped Price`. In this case the sweep bandwidth is determined by the bandwidth factors specified in the **Bandwidth factors** parameter and the number of frequency steps specified in **Number of frequency steps** parameter.

Example: `Stepped Price`

Programmatic Use
Block Parameter:`FrequencyModulation`
Type:char, string
Values:char. string
Default:`Hyperbolic`

Data Types: `char` | `string`

Coefficients of a polynomial FM — Coefficients of polynomial frequency function
`[1 0 0]` (default) | (*N*+1)-length real-valued vector

Coefficients of the polynomial frequency function, specified as a length-(*N*+1) real-valued vector. The vector represents the coefficients of an *N*-th degree polynomial. The first entry in the vector is the coefficient of the highest power *N* of the polynomial. The last entry is the coefficient of the power zero term of the polynomial.

Example: `[0.5,1,1,0.5]`

Dependencies

To enable this property, set the **Frequency modulation** parameter to `Polynomial`.

Programmatic Use
Block Parameter:`PolynomialCoefficients`
Type:double
Values:vector
Default:`[1 0 0]`

Data Types: `double`

Sweep bandwidth (Hz) — Bandwidth of nonlinear FM sweep
`100e3` (default) | positive scalar

Bandwidth of the linear FM sweep, specified as a positive scalar. Units are in Hz.

Example: `50e3`

Programmatic Use
Block Parameter:`SweepBandwidth`

Type:double
Values:positive scalar
Default:`100e3`

Sweep direction — Direction of nonlinear FM sweep
Up (default) | Down

Direction of nonlinear FM sweep as `Up` (increasing frequency) or `Down` (decreasing frequency).

Programmatic Use
Block Parameter:`SweepDirection`
Type:enum
Values: Up, Down
Default: Up

Sweep interval — FM frequency sweep interval
`Positive` (default) | `Symmetric`

FM frequency sweep interval, specified as `Positive` or `Symmetric`. If you set this parameter to `Positive`, the waveform sweeps the frequency interval between *0* and *B*, where *B* is the value of the **Sweep bandwidth** parameter. If you set this parameter value to `Symmetric`, the waveform sweeps the interval between *–B/2* and *B/2*.

Programmatic Use
Block Parameter:`SweepInterval`
Type:enum
Values: Symmetric, Positive
Default:`Positive`

Envelope function — FM signal amplitude envelope
`Rectangular` (default) | `Gaussian` | `Hamming` | `Chebyshev` | `Hann` | `Kaiser` | `Taylor`

FM signal amplitude envelope function, specified as `Rectangular`, `Gaussian`, `Hamming`, `Chebyshev`, `Hann`, `Kaiser`, or `Taylor`.

Programmatic Use
Block Parameter:`Envelope`
Type:enum
Values:`Rectangular`, `Gaussian`, `Hamming`, `Chebyshev`, `Hann`, `Kaiser`, or `Taylor`
Default:`Rectangular`

Source of Frequency Offset — Source of frequency offset
`Property` (default) | `Input port`

Source of frequency offset, specified as `Property` or `Input port`.

- When set to `Property`, the offset is determined by the value of the **Frequency Offset** parameter.
- When set to `Input port`, the offset is determined by the value of the **FreqOffset** port.

Programmatic Use
Block Parameter:`FrequencyOffsetSource`

Type:enum
Values:`Property`, `Input Port`
Default:`Property`

Frequency Offset (Hz) — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `2e3`

Dependencies

To enable this parameter set the **Source of Frequency Offset** parameter to `Input port`.

Programmatic Use
Block Parameter:`FrequencyOffset`
Type:double
Values:`scalar`
Default:`0`

Output signal format — Format of the output signal
`Pulses` (default) | `Samples`

The format of the output signal, specified as `Pulses` or `Samples`.

If you set this parameter to `Samples`, the output of the block consists of multiple samples. The number of samples is the value of the **Number of samples in output** parameter.

If you set this parameter to `Pulses`, the output of the block consists of multiple pulses. The number of pulses is the value of the **Number of pulses in output** parameter.

Programmatic Use
Block Parameter:`OutputFormat`
Type:enum
Values:`Pulses` `Samples`
Default:`Pulses`

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use
Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Number of pulses in output — Number of pulses in output
1 (default) | positive integer

Number of pulses in the block output, specified as a positive integer.

Example: 2

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Pulses`.

Programmatic Use
Block Parameter:`NumPulses`
Type:double
Values:`positive scalar`
Default:`1`

Data Types: `double`

Enable PRF Output — Enable output of PRF
`off` (default) | `on`

Select this parameter to enable the PRF output port.

Dependencies

To enable this parameter, set **Output signal format** to `Pulses`.

Programmatic Use
Block Parameter:`PRFOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Enable Matched Filter Coeficients Output — Enable output of matched filter coefficients
`off` (default) | `on`

Select this parameter to enable the `Coeff` output port.

Programmatic Use
Block Parameter:`CoefficientOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2023a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using Simulink® Coder™.

See Also
`phased.NonlinearFMWaveform`

Phase Coded Waveform

Phase-coded pulse waveform

Libraries:
Phased Array System Toolbox / Waveforms

Description

The Phase-Coded Waveform block generates samples of a phase-coded pulse waveform with specified chip width, pulse repetition frequency (PRF), and phase code. The block outputs an integer number of pulses or samples.

Ports

Input

PRFIdx — PRF Index
positive integer

Index to select the pulse repetition frequency (PRF), specified as a positive integer. The index selects the PRF from the predefined vector of values specified by the **Pulse repetition frequency (Hz)** parameter.

Example: 4

Dependencies

To enable this port, select **Enable PRF selection input**.

Data Types: `double`

FreqOffset — Frequency offset
scalar

Frequency offset in Hz, specified as a scalar.

Example: `2e3`

Dependencies

To enable this port, set **Source of Frequency Offset** to `Input port`.

Data Types: `double`

Output

Y — Pulse waveform
complex-valued vector

Pulse waveform samples, returned as a complex-valued vector.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, returned as a positive scalar.

Dependencies

To enable this port, set the **Output signal format** parameter to `Pulses` and then select the **Enable PRF output** parameter.

Data Types: `double`

Coeff — Matched filter coefficients
vector

Matched filter coefficients, returned as a vector.

Dependencies

To enable this port, select **Enable Matched Filter Coefficients Output**.

Data Types: `double`

Parameters

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. Set the ratio of the **Sample rate (Hz)** parameter to the **Pulse repetition frequency (Hz)** parameter to an integer.

- The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.
- The product of **Sample rate (Hz)** and **Chip width (s)** must be an integer. This restriction is equivalent to requiring that the chip width is an integer multiple of the sample interval.

Units are in Hz.

Example: `5e3`

Phase code — Code type used for phase modulation
`Frank` (default)

Code type used for phase modulation, specified as one of

- `Barker`
- `Frank`
- `P1`
- `P2`
- `P3`
- `P4`

- `Px`
- `Zadoff-Chu`

Example: `P2`

Chip width (s) — Chip time duration
`1e-5` (default) | positive scalar

Duration of every chip in a phase-coded waveform, specified as a positive scalar. The value of this parameter must satisfy these constraints:

- The product of **Chip width (s)**, **Number of chips**, and **Pulse repetition frequency (Hz)** must be less than or equal to one. This restriction is equivalent to requiring that the pulse length is less than the pulse repetition interval.
- The product of **Sample rate (Hz)** and **Chip width (s)** must be an integer. This restriction is equivalent to requiring that the chip width is an integer multiple of the sample interval.

Units are in seconds.

Example: `2e-4`

Number of chips — Number of chips in waveform
`4` (default) | positive integer

Number of chips in a phase-coded waveform, specified as a positive integer. The product of the **Chip width (s)**, **Number of chips**, and **Pulse repetition frequency (Hz)** parameters must be less than or equal to one. This restriction is equivalent to requiring that the chip width is an integer multiple of the sample interval.

The table shows additional constraints on the number of chips for different code types.

If the Phase code parameter is...	Then the Number of chips parameter must be...
`Frank`, `P1`, or `Px`	A perfect square
`P2`	An even number that is a perfect square
`Barker`	2, 3, 4, 5, 7, 11, or 13

Example: `9`

Zadoff-Chu sequence index — Sequence index for Zadoff-Chu code type
`1` (default) | positive integer

Sequence index for Zadoff-Chu code type, specified as a positive integer. The values of the **Zadoff-Chu sequence index** and the **Number of chips** parameters must be relatively prime.

Example: `2`

Dependencies

To enable this parameter, set **Phase Code** to `Zadoff-Chu`.

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1e4` (default) | positive scalar

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The value of **Pulse repetition frequency (Hz)** must satisfy these constraints:

- The product of **Pulse width** and **Pulse repetition frequency (Hz)** must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of **Pulse repetition frequency** must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* by using block parameter settings alone or in conjunction with the input port, `PRFIdx`.

- When the **Enable PRF selection input** parameter is not selected, set the *PRF* using block parameters.
  - To implement a constant *PRF*, specify **Pulse repetition frequency (Hz)** as a positive scalar.
  - To implement a staggered *PRF*, specify **Pulse repetition frequency (Hz)** as a row vector with positive values. After the waveform reaches the last element of the vector, the process continues cyclically with the first element of the vector. When *PRF* is staggered, the time between successive output pulses cycles through the successive values of the *PRF* vector.
- When the **Enable PRF selection input** parameter is selected, you can implement a selectable *PRF* by specifying **Pulse repetition frequency (Hz)** as a row vector with positive real-valued entries. But this time, when you execute the block, select a *PRF* by passing an index into the *PRF* vector into the `PRFIdx` port.

In all cases, the number of output samples is fixed when you set the **Output signal format** to `Samples`. When you use a varying *PRF* and set **Output signal format** to `Pulses`, the number of output samples can vary.

Programmatic Use
Block Parameter:PRF
Type:double
Values:positive scalar
Default:`1e6`

Enable PRF selection input — Select predefined PRF
off (default) | on

Select this parameter to enable the `PRFIdx` port.

- When enabled, pass in an index into a vector of predefined PRFs. Set predefined PRFs using the **Pulse repetition frequency (Hz)** parameter.
- When not enabled, the block cycles through the vector of PRFs specified by the **Pulse repetition frequency (Hz)** parameter. If **Pulse repetition frequency (Hz)** is a scalar, the PRF is constant.

Programmatic Use
Block Parameter:`PRFSelectionInputPort`

Type:logical
Values:positive scalar
Default:`off`

Source of Frequency Offset — Source of frequency offset
`Property` (default) | `Input port`

Source of frequency offset, specified as `Property` or `Input port`.

- When set to `Property`, the offset is determined by the value of the **Frequency Offset** parameter.
- When set to `Input port`, the offset is determined by the value of the **FreqOffset** port.

Programmatic Use
Block Parameter:`FrequencyOffsetSource`
Type:enum
Values:`Property`, `Input Port`
Default:`Property`

Frequency Offset (Hz) — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `2e3`

Dependencies

To enable this parameter set the **Source of Frequency Offset** parameter to `Input port`.

Programmatic Use
Block Parameter:`FrequencyOffset`
Type:double
Values:`scalar`
Default:`0`

Source of simulation sample time — Source of simulation sample time
`Derive from waveform parameters` (default) | `Inherit from Simulink engine`

Source of simulation sample time, specified as `Derive from waveform parameters` or `Inherit from Simulink engine`. When set to `Derive from waveform parameters`, the block runs at a variable rate determined by the PRF of the selected waveform. The elapsed time is variable. When set to `Inherit from Simulink engine`, the block runs at a fixed rate so the elapsed time is a constant.

Dependencies

To enable this parameter, select the **Enable PRF selection input** parameter.

Programmatic Use
Block Parameter:`SimulationTimeSource`
Type:enum
Values:`Derive from waveform parameters`, `Inherit from Simulink engine`
Default:`Derive from waveform parameters`

Output signal format — Format of the output signal
`Pulses` (default) | `Samples`

The format of the output signal, specified as `Pulses` or `Samples`.

If you set this parameter to `Samples`, the output of the block consists of multiple samples. The number of samples is the value of the **Number of samples in output** parameter.

If you set this parameter to `Pulses`, the output of the block consists of multiple pulses. The number of pulses is the value of the **Number of pulses in output** parameter.

Programmatic Use
Block Parameter:`OutputFormat`
Type:enum
Values:`Pulses` `Samples`
Default:`Pulses`

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use
Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Number of pulses in output — Number of pulses in output
`1` (default) | positive integer

Number of pulses in the block output, specified as a positive integer.

Example: `2`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Pulses`.

Programmatic Use
Block Parameter:`NumPulses`
Type:double
Values:`positive scalar`
Default:`1`

Data Types: `double`

Enable PRF Output — Enable output of PRF
`off` (default) | `on`

Select this parameter to enable the PRF output port.

Dependencies

To enable this parameter, set **Output signal format** to `Pulses`.

Programmatic Use
Block Parameter:`PRFOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Enable Matched Filter Coeficients Output — Enable output of matched filter coefficients
`off` (default) | `on`

Select this parameter to enable the `Coeff` output port.

Programmatic Use
Block Parameter:`CoefficientOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

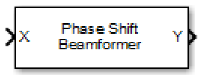
Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also
`phased.PhaseCodedWaveform`

Phase Shift Beamformer

Narrowband phase-shift beamformer



Libraries:
Phased Array System Toolbox / Beamforming

Description

The Phase Shift Beamformer block performs delay-and-sum beamforming. The delay is approximated using the phase-shift approximation in the time domain.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Ang — Beamforming direction
2-by-1 real-valued vector | 2-by-*L* real-valued matrix

Beamforming direction, specified as a 2-by-*L* real-valued matrix, where *L* is the number of beamforming directions. Each column takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Source of beamforming direction** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-*L* complex-valued matrix

Beamformed output, returned as an *M*-by-*L* complex-valued matrix. The quantity *M* is the number of signal samples and *L* is the number of desired beamforming directions specified by the `Beamforming direction` parameter or from the `Ang` port.

Data Types: `double`

W — Beamforming weights
N-by-*L* complex-valued matrix

Beamformed weights, returned as an *N*-by-*L* complex-valued matrix. The quantity *N* is the number of array elements. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays. *L* is the number of desired beamforming directions specified in the `Ang` port or by the `Beamforming direction (deg)` property. There is one set of weights for each beamforming direction.

Dependencies

To enable this port, select the **Enable weights output** checkbox.

Data Types: `double`

Parameters

Main tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Source of beamforming direction — Source of beamforming direction
`Property` (default) | `Input port`

Source of beamforming direction, specified as `Property` or `Input port`. When you set **Source of beamforming direction** to `Property`, you then set the direction using the **Beamforming direction (deg)** parameter. When you select `Input port`, the direction is determined by the input to the `Ang` port.

Beamforming direction (deg) — Beamforming directions
2-by-*L* real-valued matrix

Beamforming directions, specified as a *2*-by-*L* real-valued matrix, where *L* is the number of beamforming directions. Each column takes the form `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this parameter, set the **Source of beamforming direction** parameter to `Property`.

Number of bits in phase shifters — Number of phase shift quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Weights normalizing method — Specify weights normalization method
`Distortionless` (default) | `Preserve power`

Specify this parameter to set the weights normalizing method. Choose `Distortionless` to set the gain in the beamforming direction to zero dB. Choose `Preserve power` to set the norm of the weights to one.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-Q vector. Q must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued Q-by-P matrix | real-valued Q-by-P-by-L array

Phase of the combined antenna radiation pattern, specified as a Q-by-P matrix or a Q-by-P-by-L array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, Q equals the length of the vector specified by the **Elevation angles (deg)** parameter and P equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, Q equals the length of the vector specified by the **Theta Angles (deg)** parameter and P equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity L equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-L row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-L vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

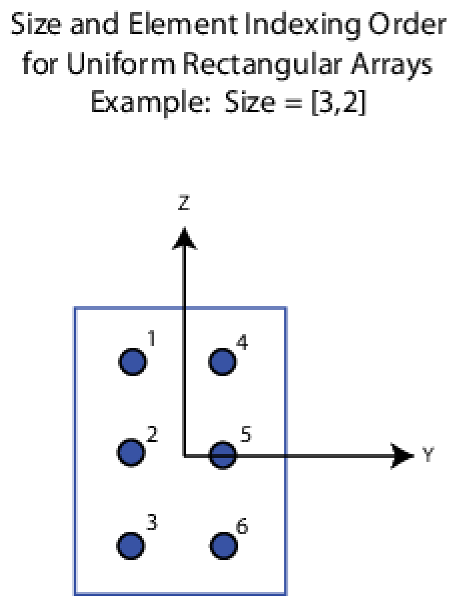
- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to URA.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to URA.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.

Array Normal Parameter Value	Element Positions and Boresight Directions
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
[0;0;0] (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position [x;y;z]of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to Conformal Array.

Element normals (deg) — Direction of conformal array element normal vectors
[0;0] | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form [azimuth;elevation] with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to Conformal Array.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

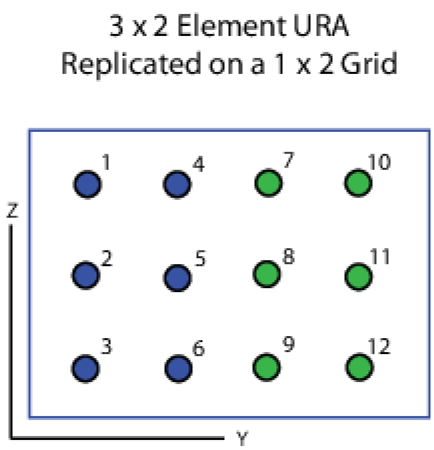
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as ULA or URA.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

Introduced in R2014b

See Also

`phased.PhaseShiftBeamformer`

Motion Platform

Motion platform

Libraries:
Phased Array System Toolbox / Environment and Target

Description

The Motion Platform block models the motion of multiple platforms such as airplanes, ground vehicles, and/or receiving and transmitting sensors arrays, determining their positions and velocities. The platforms move along trajectories determined by initial positions and velocities, according to which motion model you choose — the velocity or acceleration model. The platform positions and velocities are updated at each simulation step. In addition, you can specify initial orientations for the platforms and obtain orientation updates.

Ports

Input

Ref — Reference time input
scalar

Reference time input, specified as a scalar. Units are in seconds.

Dependencies

To enable this port, set the **Source of elapsed simulation time** parameter to `Derive from reference input port`.

Data Types: `double`

Vel — Platform velocity input
real-valued 3-by-*N* matrix

Platform velocity, specified as a real-valued 3-by-*N* matrix where *N* is the number of platforms to model. Units are meters per second.

Dependencies

To enable this port, set the **Model of object motion** parameter to `Velocity` and the **Source of velocity** parameter to `Input port`.

Data Types: `double`

Acl — Platform acceleration input
real-valued 3-by-*N* matrix

Platform acceleration input, specified as a real-valued 3-by-*N* matrix where *N* is the number of platforms to model. Units are meters per second-squared.

Dependencies

To enable this port, set the **Model of object motion** parameter to `Acceleration` and the **Source of acceleration** parameter to `Input port`.

Data Types: `double`

Output

Pos — Platform position output
real-valued 3-by-*N* matrix

Current position of platform, returned as a real-valued 3-by-1 column vector in the form of `[x;y;z]` or a real-valued 3-by-*N* matrix where *N* is the number of platforms to model. Each column takes the form `[x;y;z]`. Units are meters.

Data Types: `double`

Vel — Platform velocity output
real-valued 3-by-*N* matrix

Current velocity of platform, specify as a real-valued 3-by-1 column vector in the form of `[vx;vy;vz]` or a real-valued 3-by-*N* matrix where *N* is the number of platforms to model. Each column taking the form `[vx;vy;vz]`. Velocity units are meters per second.

Data Types: `double`

LAxes — Platform orientation output
real-valued 3-by-3-by-*N* matrix

Current platform orientation axes, returned as real-valued 3-by-3-by-*N* matrix where *N* is the number of platforms to model. Each 3-by-3 submatrix is an orthonormal matrix. This output is enabled when you set the `OrientationAxesOutputPort` property to `true`. The current platform axes rotate around the normal vector to the path of the platform.

Dependencies

To enable this port, select the **Enable orientation axes output** check box.

Data Types: `double`

Parameters

Model of object motion — Object motion model

`Velocity` (default) | `Acceleration` | `Custom`

Specify the object motion model as `Velocity`, `Acceleration`, or `Custom`. When you set this parameter to `Velocity`, the platform follows a constant velocity trajectory during each simulation step. When you set this parameter to `Acceleration`, the platform follows a constant acceleration trajectory during each simulation step. When you set the parameter to `Custom`, the platform motion follows a sequence of waypoints specified by the **Custom trajectory waypoints** parameter. The object performs a piecewise cubic interpolation on the waypoints to derive the position and velocity at each time step.

Initial position (m) — Initial position of platform

`[0;0;0]` (default)

Specify the initial position of the platform in meters as a 3-by-*N* matrix where each column represents the initial position of a platform in the form `[x;y;z]`. The quantity *N* is the number of platforms.

Initial velocity (m/s) — Initial velocity of platform

`[0;0;0]` (default)

Specify the initial velocity of the platform in m/s as a 3-by-*N* matrix where each column represents the initial velocity of a platform in the form `[vx;vy;vz]`. The quantity *N* is the number of platforms.

Dependencies

This parameter appears only when you set the **Source of velocity** or the **Source of acceleration** parameters to `Input port`.

Source of velocity — Source of velocity data

`Property` (default) | `Input port`

This parameter appears only when you set the **Model of object motion** parameter to `Velocity`. Then, you must supply velocity data for the model. Specify the **Source of velocity** data as either coming from a `Property` or an `Input port`.

Source of velocity	Use these model parameters or ports
Property	**Initial position (m)** parameter **Velocity (m/s)** parameter
`Input port`	**Initial position (m)** parameter **Initial velocity (m/s)** parameter `Vel` input port

Dependencies

This parameter appears only when you set the **Model of object motion** parameter to `Velocity`.

Velocity (m/s) — Current velocity of platform

`[0;0;0]` (default)

Specify the current velocity of the platforms in m/s as a 3-by-*N* matrix where each column represents the current velocity of a platform in the form `[vx;vy;vz]`.

Dependencies

This parameter appears only when you set the **Model of object motion** parameter to `Velocity` and set the **Source of velocity** parameter to `Property`.

Source of acceleration — Source of acceleration data

`Property` (default) | `Input port`

This parameter appears only when you set the **Model of object motion** parameter to `Acceleration`. Then, you must supply acceleration values for the model. Specify the **Source of acceleration** data as either coming from a `Property` or an `Input port`.

Source of acceleration	Use these model parameters or ports
`Property`	**Initial Position (m)** parameter **Initial Velocity (m/s)** parameter **Acceleration (m/s^2)** parameter
`Input port`	**Initial Position (m)** parameter **Initial Velocity (m/s)** parameter `Acl` input port

Dependencies

This parameter appears only when you set the **Model of object motion** parameter to `Acceleration`.

Acceleration (m/s^2) — Acceleration of platform

`[0;0;0]` (default)

Specify the current acceleration of the platforms in m/s^2 as a 3-by-*N* matrix where each column represents the current acceleration of a platform in the form `[ax;ay;az]`.

Dependencies

This parameter appears when you set the **Model of object motion** parameter to `Acceleration` and set the **Source of acceleration** parameter to `Property`.

Custom trajectory waypoints — Custom trajectory waypoints

`[ 0, 0, 0, 0, 0, 0, 0; 1, 0, 0, 0, 0, 0, 0 ]` (default)

Custom trajectory waypoints, specified as a real-valued *M*-by-*L* matrix, or *M*-by-*L*-by-*N* array. *M* is the number of waypoints. *L* is either 4 or 7.

- When *L* is 4, the first column indicates the times at which the platform position is measured. Columns 2 - 4 are position measurements in x, y, and z coordinates. The velocity is derived from the position measurements.
- When *L* is 7, columns 5 - 7 in the matrix are velocity measurements in x, y, and z coordinates.

When you set the **Custom trajectory waypoints** parameter to a three-dimensional array, the number of pages, *N*, represent the number of platforms. Time units are in seconds, position units are in meters, and velocity units are in meters per second.

Dependencies

To enable this property, set the **Model of object motion** property to `Custom`.

Mechanical scanning mode — Mechanical scanning mode

`None` (default) | `Circular` | `Sector`

Mechanical scan mode for platform, specified as `None`, `Circular`, or `Sector`, where `None` is the default. When you set the **Mechanical scanning mode** parameter to `Circular`, the platform scan clockwise 360 degrees continuously in the azimuthal direction of the platform orientation axes. When you set the **Mechanical scanning mode** parameter to `Sector`, the platform scans clockwise in the azimuthal direction in the platform orientation axes within a range specified by the **Azimuth scan angle span (deg)** parameter. When the platform scan reaches the span limits, the scan reverses direction and scans back to the other scan limit. Scanning happens within the orientation axes of the platform.

Initial scan angle (deg) — Initial scan angle of platform

`0` (default)

Initial scan angle of platform, specified as a 1-by-*N* vector where *N* is the number of platforms. The scanning occurs in the local coordinate system of the platform. The **Initial orientation axes** parameter specifies the original local coordinate system. At the start of the simulation, the orientation axes specified by the **Initial orientation axes** are rotated by the angle specified in the `InitialScanAngle`**Initial scan angle (deg)** parameter. The default value is zero. Units are in degrees.

Dependencies

This parameter applies when you set the **Mechanical scanning mode** parameter to `Circular` or `Sector`.

Azimuth scan angle span (deg) — Azimuth span

`[-60,60]` (default)

Specify the azimuth angle span as an *N*-by-2 matrix where *N* is the number of platforms. Each row of the matrix specifies the scan range of the corresponding platform in the form `[ScanAngleLowerBound ScanAngleHigherBound]`. The default value is `[-60 60]`. Units are in degrees.

Dependencies

To enable this parameter, set the **Mechanical scanning mode** parameter to `Sector`.

Azimuth scan rate (deg/s) — Azimuth scan rate

`10` (default)

Specify the azimuth scan rate as a 1-by-*N* vector where *N* is the number of platforms. Each entry in the vector is the azimuth scan rate for the corresponding platform. The default value is 10 degrees/second. Units are in degrees/second.

Dependencies

To enable this parameter, set the **Mechanical scanning mode** parameter to `Circular` or `Sector`.

Initial orientation axes — Initial orientation axes of platform

`eye(3)` (default)

Specify the three axes that define the initial local *(x,y,z)* coordinate system at the platform as a 3-by-3-by-*N* matrix. Each column of the matrix represents an axis of the local coordinate system. The three axes must be orthonormal.

Enable orientation axes output — Output orientation axes

off (default) | on

Select this check box to obtain the instantaneous orientation axes of the platform via the output port `LAxes`. The port appears only when the check box is selected.

Source of elapsed simulation time — Source of elapsed simulation time

`Auto` (default) | `Derive from reference input port`

Specify the source for elapsed simulation time as `Auto` or `Derive from reference input port`. When you choose `Auto`, the block computes the elapsed time. When you choose `Derive from reference input port`, the block uses the time duration of a reference signal passed into the `Ref` input port.

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

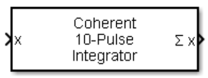
Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also
`phased.Platform`

Pulse Integrator

Coherent or noncoherent pulse integration



Libraries:
Phased Array System Toolbox / Detection

Description

The Pulse Integrator block performs coherent or noncoherent integration of successive pulses of a signal and puts out an integrated output. You can specify how many pulses to integrate and the number of overlapped pulses in successive integrations.

Ports

Input

x — Pulse input data
matrix

Pulse input data, specified as a matrix. Each column of `x` is one pulse.

Data Types: `double`

Output

∑x — Integrated pulse
column vector

Integrated pulse, returned as an N-by-1 column vector, where N is the number of rows in the input `x`.

Data Types: `double`

Parameters

Integration method — Integration method

`Coherent` (default) | `Noncoherent`

Specify the integration method as `Coherent` or `Noncoherent`.

Number of pulses to integrate — Number of pulses to integrate

`10` (default)

Specify the number of pulses to integrate as an integer.

Integration overlap (in pulses) — Integration overlap

`0` (default)

Specify the number of overlapped pulses in successive integrations as an integer. This number must be less than the value specified in **Number of pulses to integrate**.

Version History

Introduced in R2014b

See Also

`pulsint`

Radar Target

Radar target

Libraries:
Phased Array System Toolbox / Environment and Target

Description

The Radar Target block models a radar target that reflects the signal according to the specified radar cross section (RCS). The block supports a non-fluctuating target model and all four Swerling target models. The block does not support polarized signals.

Ports

Input

X — Incident signal
complex-valued *N*-by-*1* column vector | complex-valued *N*-by-*M* matrix

Incident signal, specified as a complex-valued *N*-by-*1* column vector or complex-valued *N*-by-*M* matrix. The value *M* is the number of signals. Each signal corresponds to a different target. The value *N* is the number of samples in each signal.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

RCS — Mean radar cross-section
nonnegative scalar | 1-by-*M* row vector of nonnegative values

Mean radar cross-section, specified as nonnegative scalar or 1-by-*M* row vector of nonnegative values. *M* is the number of targets. This parameter applies to fluctuating and non-fluctuating targets. If you specify *M* incident signals, you can specify the radar cross-section as a scalar or as a 1-by-*M* vector. For a scalar, the same value will be applied to all signals. Units are in m^2.

Dependencies

To enable this port, set the **Source of mean radar cross section** pull down menu to `Input port`.

Data Types: `double`

Update — Indicator to update radar cross-section
0 | 1

Indicator to update radar cross-section for fluctuating targets, specified as 0 or 1. When set to 0, the radar cross-section is constant. When set to 1, the radar cross-section fluctuates.

Dependencies

To enable this port, set the **Fluctuation model** pull down menu to anything except `Nonfluctuating`.

Data Types: `double`

Output

Out — Reflected signal
complex-valued *N*-by-*1* column vector | complex-valued *N*-by-*M* matrix

Reflected signal, returned as a complex-valued *N*-by-*1* column vector or complex-valued *N*-by-*M* matrix. The value *M* is the number of signals. Each signal corresponds to a different target. The value *N* is the number of samples in each signal.

Parameters

Source of mean radar cross section — Source of mean radar cross section

`Property` (default) | `Input port`

Specify whether the target’s mean radar cross-section (RCS) value comes from the **Mean radar cross section** parameter of this block or from an input port. Values of this parameter are

`Property`	The **Mean radar cross section** parameter for this block specifies the mean RCS value.
`Input port`	Choosing this value creates the RCS input port to specify the mean radar cross-section.

Mean radar cross section (m^2) — Mean radar cross section

`1`

Specify the mean value of the target's radar cross section, in square meters, as a nonnegative scalar. This parameter appears only when the **Source of mean radar cross section** parameter is set to `Property`.

Fluctuation model — Fluctuation model

`Nonfluctuating` (default) | `Swerling1` | `Swerling2` | `Swerling3` | `Swerling4`

Specify the statistical model of the target as one of `Nonfluctuating`, `Swerling1`, `Swerling2`, `Swerling3`, or `Swerling4`. Setting this parameter to a value other than `Nonfluctuating`, allows setting cross-sections parameters via an input port, `Update`.

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`.

Data Types: `double`

Operating frequency (Hz) — Operating frequency

3e8 (default) | positive real scalar

Specify the carrier frequency of the signal that reflects from the target, as a positive scalar in hertz.

Simulate using — Block simulation method
Interpreted Execution (default) | Code Generation

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

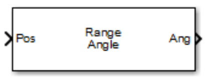
Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also
`phased.RadarTarget`

Range Angle Calculator

Range and angle calculations



Libraries:
Phased Array System Toolbox / Environment and Target

Description

The Range Angle Calculator block calculates the range and/or the azimuth and elevation angles of several positions with respect to a reference position and with respect to a reference axes orientation. The reference position and reference axes can be specified in the block dialog or passed in through input ports.

Ports

Input

Pos — Source point position
real-valued 3-by-1 vector | real-valued 3-by-*N* matrix

Source point position, specified as a real-valued 3-by-1 vector or a real-valued 3-by-*N* matrix where *N* is the number of source points. Each column contains the Cartesian coordinates of a point in the form `[x;y;z]`.

When `Pos` is a 3-by-*N* matrix, you must specify `RefPos` as a 3-by-*N* matrix for *N* reference positions. If all the reference points are identical, you can specify `RefPos` by a single 3-by-1 vector. Units are in meters.

Example: [1000;2000;50]

Data Types: `double`

RefPos — Reference point position
`[0;0;0]` (default) | real-valued 3-by-1 vector | real-valued 3-by-*N* matrix

Reference point position, specified as a real-valued 3-by-1 vector or a real-valued 3-by-*N* matrix where *N* is the number of reference points. Each column contains the Cartesian coordinates of point in the form `[x;y;z]`.

When `RefPos` is a 3-by-*N* matrix, you must specify `Pos` as a 3-by-*N* matrix for *N* source positions. If all the source points are identical, you can specify `Pos` by a single 3-by-1 vector. Reference position units are meters.

Example: [100;100;10]

Dependencies

To enable this port, set the **Reference position source** drop down menu to `Input port`.

Data Types: `double`

RefAxes — Local coordinate system axes
[1 0 0;0 1 0;0 0 1] (default) | real-valued 3-by-3 vector | real-valued 3-by-3-by-*N* array

Local coordinate system axes, specified as a real-valued 3-by-3 matrix or a 3-by-3-by-*N* array. For an array, each page corresponds to a local coordinate axes at each reference point. The columns in `RefAxes` specify the direction of the coordinate axes for the local coordinate system in Cartesian coordinates. *N* must match the number of columns in `Pos` or `RefPos` when these dimensions are greater than one.

Example: [100;100;10]

Dependencies

To enable this port, set the **Reference axes source** drop down menu to `Input port`.

Data Types: `double`

Output

Ang — Azimuth and elevation angles
real-valued 2-by-*N* matrix | real-valued 2-by-2*N* matrix

Azimuth and elevation angles in degrees, returned as a 2-by-*N* matrix or 2-by-2*N* matrix. Each column represents a direction angle in the form [azimuth;elevation].

When **Propagation model** is set to **Free space**, `Ang` is a 2-by-*N* matrix and represents the angle of the path from a source point to a reference point.

When **Propagation model** is set to **Two-ray**, `Ang` is a 2-by-2*N* matrix. Alternate columns of `Ang` refer to the line-of-sight path and reflected path, respectively.

Dependencies

To enable this port, set the **Output(s)** drop down menu to `Angle` or `Range and Angle`.

Data Types: `double`

Range — Propagation range
real-valued 1-by-*N* vector | real-valued 1-by-2*N* vector

Propagation range, returned as a real-valued 1-by-*N* vector or real-valued 1-by-2*N* vector.

When **Propagation model** is set to **Free space**, the size of **Range** is 1-by-*N*. The propagation range is the length of the direct path from the position defined in `Pos` to the corresponding reference position defined in `RefPos`.

When **Propagation model** is set to **Two-ray**, **Range** contains the ranges for the direct path and the reflected path. Alternate columns of **Range** refer to the line-of-sight path and reflected path, respectively for the same source-reference point pair.

Dependencies

To enable this port, set the **Output(s)** drop down menu to `Range` or `Range and Angle`.

Data Types: `double`

Parameters

Propagation model — Propagation model

`Free space` (default) | `Two-ray`

Specify the propagation model by setting this parameter to `Free space` or `Two-ray`.

Reference position source — Reference position source

`Property` (default) | `Input port`

Specify the reference position source by setting this parameter to `Property` or `Input port`. If **Reference position source** is set to `Property`, set the position using the **Reference position** parameter. If **Reference position source** is set to `Input port`, use the input port labeled `RefPos`.

Reference position — Reference position

`[0;0;0]` (default)

Specify the reference position as a 3-by-1 vector of rectangular coordinates in meters in the form `[x;y;z]`. The reference position serves as the origin of the local coordinate system. Ranges and angles of the input positions are measured with respect to the reference position. This parameter appears only when **Reference position source** is set to `Property`.

Reference axes source — Reference axes source

`Property` (default) | `Input port`

Specify the reference axes source by setting this parameter to `Property` or `Input port`. If **Reference axes source** is set to `Property`, set the axes using the **Reference axes** parameter. If **Reference axes source** is set to `Input port`, use the input port labeled `RefAxes`.

Reference axes — Reference axes

`eye(3)`

Specify the reference axes of the local coordinate system with which to calculate range and angles in the form of a 3-by-3 orthonormal matrix. Each column of the matrix specifies the direction of an axis for the local coordinate system in the form of `[x; y; z]` with origin at the reference position. This parameter appears only when **Reference axes source** is set to `Property`.

Output(s) — Output(s)

`Angle` (default) | `Range` | `Range and Angle`

Specify the desired output(s) of the block. Each type of output is sent to a different port depending on the parameter value.

Value	Port
`Angle`	`Ang`
`Range`	`Range`

Value	Port
Range and Angle	Ang and Range

Simulate using — Block simulation method
Interpreted Execution (default) | Code Generation

Block simulation, specified as Interpreted Execution or Code Generation. If you want your block to use the MATLAB interpreter, choose Interpreted Execution. If you want your block to run as compiled code, choose Code Generation. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using Code Generation. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in Accelerator mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	Normal	Accelerator	Rapid Accelerator
Interpreted Execution	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
Code Generation	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter: SimulateUsing
Type:enum
Values: Interpreted Execution, Code Generation
Default: Interpreted Execution

Version History

Introduced in R2014b

See Also

rangeangle

Range-Angle Response

Obtain range-angle response map for array

Libraries:
Phased Array System Toolbox / Detection

Description

The Range-Angle Response block computes the range-angle map of an input signal. The output response is a matrix or a three-dimensional array whose rows represent range gates and columns represent angles. Pages represent

Ports

Input

X — Input signal data cube
complex-valued *K*-by-*N* matrix | complex-valued *K*-by-*N*-by-*L* array

Input signal cube, specified as a complex-valued *K*-by-*N* matrix or complex-valued *K*-by-*N*-by-*L* array. The contents of the data cube depend on the type of range-angle processing specified by the different syntaxes.

- *K* is the number of fast-time or range samples.
- *N* is the number of independent spatial channels such as sensors or directions.
- *L* is the slow-time dimension that corresponds to the number of pulses or sweeps in the input signal.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency

Dependencies

To enable this input argument, set the value of **Range processing method** to `FFT` and do not select the **Dechirp input signal** check box.

Data Types: `double`

Xref — Reference signal used for dechirping
complex-valued *K*-by-1 column vector

Reference signal used for dechirping, specified as a complex-valued *K*-by-1 column vector. The number of rows must equal the length of the fast-time dimension of `X`.

Dependencies

To enable this input argument, set the value of **Range processing method** to `FFT` and select the **Dechirp input signal** check box.

Data Types: `double`

Coeff — Matched filter coefficients
complex-valued *P*-by-1 column vector

Matched filter coefficients, specified as a complex-valued *P*-by-1 column vector. *P* must be less than or equal to *K*. *K* is the number of fast-time or range sample.

Dependencies

To enable this input argument, set the value of **Range processing method** to `Matched filter`.

Data Types: `double`

El — Elevation angle
scalar

Elevation angle of response, specified as a scalar between –90° and +90°. The range-angle response is computed for this elevation. Units are in degrees.

Dependencies

To enable this argument, set the **Source of elevation angle** parameter to `Input port`.

Data Types: `double`

Output

Resp — Range response data cube
complex-valued *M*-element column vector | complex-valued *M*-by-*L* matrix | complex-valued *M*-by-*N* by-*L* array

Range response data cube, returned as one of the following:

- Complex-valued *M*-element column vector
- Complex-valued *M*-by-*L* matrix
- Complex-valued *M*-by-*N* by-*L* array

The value of *M* depends on the type of processing

Range Processing Method	Value of *M*
`FFT`	If you set the **Source of FFT length in range processing** parameter to `Auto`, then *M* = *K*, the length of the fast-time dimension of `X`. Otherwise, *M* equals the value of the FFT length in range processing parameter.
`Matched filter`	*M* = *K*, the length of the fast-time dimension of `X`.

Data Types: `double`

Range — Range values along range dimension
real-valued *M*-by-1 column vector

Range values along range dimension, returned as a real-valued *M*-by-1 column vector. This vector defines the ranges that correspond to the fast-time dimension of the `RESP` output data cube. *M* is the length of the fast-time dimension of `RESP`. Range values are monotonically increasing and equally spaced. Units are in meters.

Data Types: `double`

Ang — Angle values along angle direction
P-by-1 real-valued vector

Angle values corresponding to the samples along angle direction, returned as a *P*-by-1 real-valued vector. Units are in degrees.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Range processing method — Range processing method
`Matched filter` (default) | `FFT`

Range processing method, specified as `Matched filter` or `FFT`.

- `Matched filter` — The object match-filters the incoming signal. This approach is commonly used for pulsed signals, where the matched filter is the time reverse of the transmitted signal.
- `FFT` — The object applies an FFT to the input signal. This approach is commonly used for chirped signals such as FMCW and linear FM pulsed signals.

Data Types: `char`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

FM sweep slope (Hz/s) — Linear FM sweep slope
`1.0e9` (default) | scalar

Linear FM sweep slope, specified as a scalar. The fast-time dimension of the X input port must correspond to sweeps having this slope.

Example: `1.5e9`

Dependencies

To enable this parameter, set the **Range processing method** parameter to `FFT`.

Data Types: `double`

Dechirp input signal — Enable dechirping of input signals
`on` (default) | `off`

Option to enable dechirping of input signals, specified as `on` or `off`. Not selecting this check box indicates that the input signal is already dechirped and no dechirp operation is necessary. Select this check box when the input signal requires dechirping.

Dependencies

To enable this parameter, set the **Range processing method** parameter to `FFT`.

Data Types: `Boolean`

Source of FFT length in range — Source of FFT length for range processing of dechirped signals
`Auto` (default) | `Property`

Source of the FFT length used for the range processing of dechirped signals, specified as `Auto` or `Property`.

- `Auto` — The FFT length equals the length of the fast-time dimension of the input data cube.
- `Property` — Specify the FFT length by using the **FFT length in range processing** parameter.

Dependencies

To enable this parameter, set the **Range processing method** parameter to `FFT`.

Data Types: `char`

FFT length in range processing — FFT length used for range processing
`1024` (default) | positive integer

FFT length used for range processing, specified as a positive integer.

Dependencies

To enable this parameter, set the **Range processing method** parameter to `FFT` and the **Source of FFT length in range processing** parameter to `Property`.

Data Types: `double`

Range processing window — FFT weighting window for range processing
`None` (default) | `Hamming` | `Chebyshev` | `Hann` | `Kaiser` | `Taylor`

FFT weighting window for range processing, specified as `None`, `Hamming`, `Chebyshev`, `Hann`, `Kaiser`, or `Taylor`.

If you set this parameter to `Taylor`, the generated Taylor window has four nearly constant sidelobes next to the mainlobe.

Dependencies

To enable this parameter, set the **Range processing method** parameter to `FFT`.

Data Types: `char`

Range sidelobe attenuation level — Sidelobe attenuation for range processing
`30` (default) | scalar

Sidelobe attenuation for range processing, specified as a positive scalar. This attenuation applies only to Kaiser, Chebyshev, or Taylor windows. Units are in dB.

Dependencies

To enable this parameter, set the **Range processing method** parameter to `FFT` and the **Range processing window** parameter to `Kaiser`, `Chebyshev`, or `Kaiser`.

Set reference range at center — Set reference range at center of range grid
`on` (default) | `off`

Set reference range at center of range grid, specified as `on` or `off`. Selecting this check box enables you to set the reference range at the center of the range grid. Otherwise, the reference range corresponds to the beginning of the range grid.

Dependencies

To enable this parameter, set the **Range processing method** to `FFT`.

Data Types: `Boolean`

Reference range (m) — Reference range of range grid
`0.0` (default) | nonnegative scalar

Reference range of the range grid, specified as a nonnegative scalar.

- If you set the **Range processing method** parameter to `'Matched filter'`, the reference range is set to the start of the range grid.
- If you set the **Range processing method** parameter to `FFT`, the reference range is determined by the **Set reference range at center** parameter.

  - When you select the **Set reference range at center** check box, the reference range is set to the center of the range grid.
  - Otherwise, the reference range is set to the start of the range grid.

  Units are in meters.

Example: `1000.0`

Data Types: `double`

Source of elevation angle — Source of elevation angle
`Property` (default) | **Input port**

Source of elevation angle, specified as `Property` or **Input port**.

`Property`	The elevation angle comes from the **Elevation angle (deg)** parameter.
`Input port`	The elevation angle comes from an input port.

Elevation angle (deg) — Elevation angle used to calculate range-angle response
`0` (default) | scalar

Elevation angle used to calculate range-angle response, specified as a scalar. The angle must be between --90 and 90 degrees. This property applies when you set the ElevationAngleSource property to `'Property'`. The default value of this property is 0.

Angle span (deg) — Angle response span
`[-90 90]` (default) | real-valued 1-by-2 vector

Angle response span, specified as a real-valued 2-by-1 vector. The object calculates the range-angle response within the angle range, `[min_angle max_angle]`.

Example: `[-45 45]`

Data Types: `12wqqqq`` | qdouble`

Number of angle bins — Number of samples in angle span
positive integer greater than two

Number of samples in angle span used to calculate range-angle response, specified as a positive integer greater than two.

Example: `[256]`

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.

- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a Q-by-P matrix or a Q-by-P-by-L array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, Q equals the length of the vector specified by the **Elevation angles (deg)** parameter and P equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, Q equals the length of the vector specified by the **Theta Angles (deg)** parameter and P equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity L equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued Q-by-P matrix | real-valued Q-by-P-by-L array

Phase of the combined antenna radiation pattern, specified as a Q-by-P matrix or a Q-by-P-by-L array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, Q equals the length of the vector specified by the **Elevation angles (deg)** parameter and P equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, Q equals the length of the vector specified by the **Theta Angles (deg)** parameter and P equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity L equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the x-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
2 for ULA arrays and 5 for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

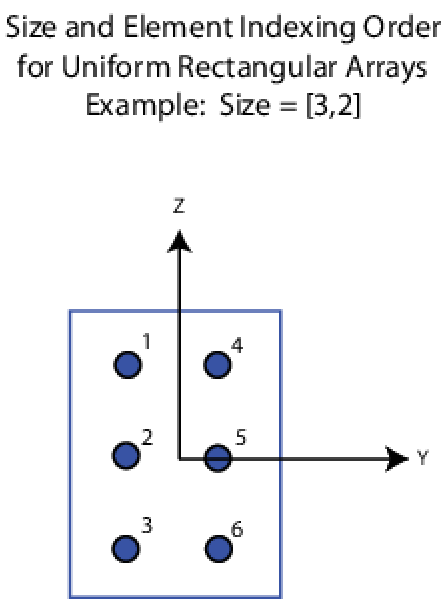
- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to URA.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to URA.

Array normal — Array normal direction
x for URA arrays or z for UCA arrays (default) | y

Array normal direction, specified as x, y, or z.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
[0;0;0] (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position [x;y;z]of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to Conformal Array.

Element normals (deg) — Direction of conformal array element normal vectors
[0;0] | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form [azimuth;elevation] with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- None
- Phase
- Time
- Custom

Selecting Phase or Time opens the Steer input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting Custom opens the WS input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to Partitioned array or Replicated subarray.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
3.0e8 (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to Partitioned array or Replicated subarray and set **Subarray steering method** to Phase.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
0 (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to Partitioned array or Replicated subarray and set **Subarray steering method** to Phase.

Subarrays layout — Subarray position specification
Rectangular (default) | Custom

Specify the layout of replicated subarrays as Rectangular or Custom.

- When you set this parameter to Rectangular, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to Custom, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

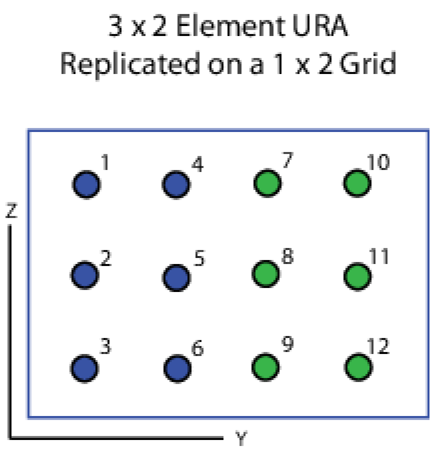
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History
Introduced in R2018b

See Also
`phased.RangeAngleResponse`

Range Doppler Response

Range-Doppler response

Libraries:
Phased Array System Toolbox / Detection

Description

The Range-Doppler Response block computes the range-Doppler map of an input signal. The output response is a matrix whose rows represent range gates and whose columns represent Doppler bins.

Ports

Input

X — Input
complex-valued *K*-by-*L* matrix | *K*-by-*N*-by-*L* array

Input data, specified as a complex-valued *K*-by-*L* matrix or *K*-by-*N*-by-*L* array where

- *K* denotes the number of fast-time samples.
- *N* denotes the number of channels such as beams or sensors. When *N* is one, only a single data channel is present.
- *L* denotes the number of pulses for matched-filter processing and the number of sweeps for FFT processing.

Data Types: `single` | `double`

Coeff — Matched filter coefficients
column vector

Matched filter coefficients, specified as a column vector.

Dependencies

To enable this port, set the **Range processing method** to `Matched filter`.

Data Types: `single` | `double`

XRef — Reference signal
vector

Reference signal, specified as a

Dependencies

To enable this port, set the **Range processing method** to `FFT` and then select the **Dechirp input signal** check box.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency, specified as a positive scalar. `prf` must be less than or equal to the sample rate specified in the `SampleRate` property divided by the length of the first dimension of the input signal, `x`. You can specify this argument as single or double precision.

Dependencies

To enable this port, set the **Source of pulse repetition frequency** drop down menu to `Input port`.

Data Types: `double`

Output

Resp — Range-Doppler response
complex

Range-Doppler, returned as a complex-valued *M*-by-*P* matrix or an *M*-by-*N*-by-*P* array.

Range — Range samples
vector

Range samples at which the range-Doppler response is evaluated. The output is a column vector of length *M*.

Data Types: `double`

Dop — Doppler or speed samples
vector

Doppler samples or speed samples at which the range-Doppler response is evaluated. returned as a column vector of length *P*. Whether `Dop` contains Doppler or speed samples depends on the **Doppler output** parameter.

Data Types: `double`

Parameters

Range processing method — Range processing method

`Matched filter` (default) | `FFT`

Specify the method of range processing as `Matched filter` or `FFT`

`Matched filter`	Applies a matched filter to the incoming signal. This technique is commonly used for pulsed signals, where the matched filter is the time reverse of the transmitted signal. Choosing this option creates the `Coeff` input port.
`FFT`	Performs range processing by applying an FFT to the input signal. This approach is commonly used with FMCW and linear FM pulsed signals.

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`.

Data Types: `double`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Reference range (m) — Reference range

`0` (default)

Specify the reference range of the range grid as a nonnegative scalar.

- If you set the **Range processing method** parameter to `Matched filter`, the reference range is set to the start of the range grid.
- If you set the **Range processing method** property to `FFT`, the reference range depends on the **Set reference range at center** check box.
  - When you select the **Set reference range at center** check box, the reference range is set to the center of the range grid.
  - If you do not select the **Set reference range at center** check box, the reference range is set to the start of the range grid.

  Units are in meters.

Source of pulse repetition frequency — Source of pulse repetition frequency

`Auto` (default) | `Property` | `Input port`

Specify the source of pulse repetition frequency as

- `Auto` — automatically compute the pulse repetition frequency (PRF). The PRF is the sample rate of the signal divided by the number of rows in the input port signal, `X`.
- `Property`— specify the pulse repetition frequency using the `PRF` parameter.

- `Input port`— specify the PRF using the PRF input port.

Use the `Property` or `Input port` option when the pulse repetition frequency cannot be determined by the signal duration, as is the case with range-gated data.

Pulse repetition frequency of the input signal (Hz) — Pulse repetition frequency of the input signal

`10e3` (default)

Specify the pulse repetition frequency of the input signal as a positive scalar. PRF must be less than or equal to the sample rate divided by the number of rows of the input signal. When the signal length is variable, use the maximum possible number of rows of the input signal instead.

Dependencies

To enable this parameter, set the **Source of pulse repetition frequency** parameter to `Property`.

Source of FFT length in Doppler processing — Source of FFT length in Doppler processing

`Auto` (default) | `Property`

Specify how the block determines the length of the FFT used in Doppler processing. Values of this parameter are

`Auto`	The FFT length equals the number of rows of the input signal.
`Property`	The **FFT length in Doppler processing** parameter of this block specifies the FFT length.

FFT length in Doppler processing — FFT length in Doppler processing

`1024`

Specify the length of the FFT used in Doppler processing as a positive integer.

Dependencies

This parameter appears only when you set **Source of FFT length in Doppler processing** to `Property`.

Doppler processing window — Doppler processing window

`None` (default) | `Hamming` | `Chebyshev` | `Hann` | `Kaiser` | `Taylor`

Specify the window used for Doppler processing using one of

`None`
`Hamming`
`Chebyshev`
`Hann`
`Kaiser`
`Taylor`

If you set this parameter to `Taylor`, the generated Taylor window has four nearly-constant sidelobes adjacent to the mainlobe.

Doppler sidelobe attenuation level — Doppler sidelobe attenuation level

`30` (default)

Specify the sidelobe attenuation level as a positive scalar, in decibels.

Dependencies

This parameter appears only when **Doppler processing window** is set to `Kaiser`, `Chebyshev`, or `Taylor`.

Doppler output — Doppler output

`Frequency` (default) | `Speed`

Specify the Doppler domain output as `Frequency` or `Speed`

`Frequency`	Doppler shift, in hertz.
`Speed`	Radial speed corresponding to Doppler shift, in meters per second.

Signal carrier frequency (Hz) — Signal carrier frequency
`3e8` (default) | positive real-valued scalar

Signal carrier frequency, specified as a positive real-valued scalar. Units are in hertz.

Data Types: `double`

FM sweep slope (Hz/s) — FM sweep slope

`1e9` (default)

Specify the slope of the linear FM sweeping, in hertz per second, as a scalar.

Dependencies

This parameter appears only when you set **Range processing method** to `FFT`.

Dechirp input signal — Dechirp input signal

off (default) | on

Select this check box to make the block perform the dechirp operation on the input signal. Clear this check box to indicate that the input signal is already dechirped and no dechirp operation is necessary.

Dependencies

This check box appears only when you set **Range processing method** to `FFT`.

Source of FFT length in range processing — Source of FFT length in range processing

`Auto` (default) | `Property`

Specify how the block determines the FFT length in range processing. Values of this parameter are

`Auto`	The FFT length equals the number of rows of the input signal.
`Property`	The FFT length is specified by **FFT length in range processing** .

Dependencies

This parameter appears only when you set **Range processing method** to `FFT`.

FFT length in range processing — FFT length in range processing

`1024` (default)

Specify the FFT length in the range domain as a positive integer.

Dependencies

This parameter appears only when you set **Range processing method** to `FFT` and **Source of FFT length in range processing** to `Property`.

Range processing window — Range processing window

`None` (default) | `Hamming` | `Chebyshev` | `Hann` | `Kaiser` | `Taylor`

Specify the window used for range processing using one of

`None`
`Hamming`
`Chebyshev`
`Hann`
`Kaiser`
`Taylor`

If you set this parameter to `Taylor`, the generated Taylor window has four nearly-constant sidelobes adjacent to the mainlobe.

Dependencies

This parameter appears only when you set **Range processing method** to `FFT`.

Set reference range at center — Set reference range at center

on (default) | off

Set reference range at the center of range grid, specified as `on` or `off`. Selecting this check box, enables you to set the reference range at the center of the range grid. Otherwise, the reference range is set to the beginning of the range grid.

Range sidelobe attenuation level — Range sidelobe attenuation level

`30` (default)

Specify the sidelobe attenuation level as a positive scalar, in decibels.

Dependencies

This parameter appears only when you set **Range processing method** to `FFT` and **Range processing window** to `Kaiser`, `Chebyshev`, or `Taylor`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><th rowspan="2">Block Simulation</th><th colspan="3">Simulation Behavior</th></tr>
<tr><th><code>Normal</code></th><th><code>Accelerator</code></th><th><code>Rapid Accelerator</code></th></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

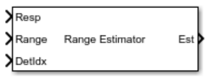
Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also
`phased.RangeDopplerResponse`

Range Estimator

Range estimation



Libraries:
Phased Array System Toolbox / Detection

Description

The Range Estimator block estimates the range of target detections obtained from the radar response data.

Ports

Input

Resp — Range-processed response data cube
complex-valued P-by-1 column vector | complex-valued M-by-P matrix | complex-valued M-by-N-by-P matrix

Range-processed response data cube, specified as a complex-valued P-by-1 column vector, a complex-valued M-by-P matrix, or a complex-valued M-by-N-by-P array. M represents the number of range samples, N is the number of sensor elements or beams, and P is the number of Doppler bins.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Range — Range grid values along range dimension
real-valued M-by-1 column vector

Range grid values along the range dimension of the data cube input, **Resp**, specified as a real-valued M-by-1 column vector. Range values must be monotonically increasing and equally spaced. Units are in meters.

Example: `[-0.3,-0.2,-0.1,0,0.1,0.2,0.3]`

Data Types: `double`

DetIdx — Detection indices
real-valued N_d-by-Q matrix

Detection indices, specified as a real-valued N_d-by-Q matrix. Q is the number of detections and N_d is the number of dimensions in the response data cube, **Resp**. Each column of **DetIdx** contains the indices of a detection in the response data cube.

NoisePower — Noise power at detection locations
positive scalar | real-valued 1-by-Q row vector of positive values

Noise power at detection locations, specified as a positive scalar or real-valued 1-by-*Q* row vector positive values. *Q* is the number of detections specified in the **DetIdx** input port.

Dependencies

To enable this port, select the **Output variance for parameter estimates** parameter, and then set **Source of noise power parameter** to `Input port`.

Clusters — Cluster IDs
real-valued 1-by-*Q* row vector of positive values

Cluster IDs, specified as a real-valued 1-by-*Q* row vector, where *Q* is the number of detections specified in the **DetIdx** input port. Each element of **Clusters** corresponds to an element of **DetIdx**.

Dependencies

To enable this input port, select the **Enable cluster ID input** checkbox.

Output

Est — Range estimate
real-valued *K*-by-1 column vector

Range estimates, specified as a real-valued *K*-by-1 column vector.

- When **Enable cluster ID input** is not selected, each range estimate corresponds to one of the columns of the **DetIdx** input port. Then *K* equals the column dimension, *Q*, of **DetIdx**.
- When **Enable cluster ID input** is selected, each range estimate corresponds to one of the cluster IDs in the **Clusters** input port. Then *K* equals the number of unique cluster IDs.

Var — Range estimation variance
positive, real-valued *K*-by-1 column vector

Range estimation variance, returned as a positive, real-valued *K*-by-1 column vector, where *K* is the dimension of **Est**. Each element of **Var** corresponds to an element of **Est**. The estimator variance is computed using the Ziv-Zakai bound.

Dependencies

To enable this output port, select the **Output variance for parameter estimates** parameter.

Parameters

Maximum number of estimates — Maximum number of estimates to report
`1` (default) | positive integer

The maximum number of estimates to report, specified as a positive integer. When the number of requested estimates is greater than the number elements in **DetIdx**, the remainder is filled with `NaN`.

Data Types: `double`

Enable cluster ID input — Enable cluster ID input
`off` (default) | `on`

Enable the **Cluster** input port to pass in cluster association information.

Data Types: `Boolean`

Output variance for parameter estimates — Enable output variance port
`off` (default) | `on`

Enables the output of the parameter estimate variances via the **Var** port.

Data Types: `Boolean`

Root-mean-square range resolution — Range resolution
`2` (default) | positive scalar

Root-mean-square range resolution of the detection, specified as a positive scalar. This parameter must have the same units as the **Range** input port.

Dependencies

To enable this parameter, select the `Output variance for parameter estimates` parameter.

Data Types: `double`

Source of noise power — Source of noise power values
`Property` (default) | `Input port`

Source of the noise power, specified as `Property` or `Input port`. If you set this parameter to `Property`, use the **Noise power** parameter to set the noise power at the detection locations. When set the parameter to `Input port`, specify noise power via the `NoisePower` input port.

Noise power — Noise power values
`1.0` (default) | positive scalar

Noise power for detections, specified as a positive scalar. The same noise power value is applied to all detections. Noise power is in linear units.

Dependencies

To enable this parameter, select the **Output variance for parameter estimates** checkbox and set the **Source of noise power** parameter to `Property`.

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2017a

See Also

Blocks
CFAR Detector | 2-D CFAR Detector | Range Doppler Response | Range Response

Objects
`phased.CFARDetector` | `phased.CFARDetector2D` | `phased.RangeEstimator` | `phased.RangeResponse` | `phased.RangeDopplerResponse`

Range Response

Range response

Libraries:
Phased Array System Toolbox / Detection

Description

The Range Response block performs range filtering on fast-time (range) data, using either a matched filter or an FFT-based algorithm. The output is typically used as input to a detector. Matched filtering improves the SNR of pulsed waveforms. For continuous FM signals, FFT processing extracts the beat frequency of FMCW waveforms. Beat frequency is directly related to range.

The input to the block is a radar data cube. The organization of the data cube follows the Phased Array System Toolbox convention. The first dimension of the cube represents the fast time samples or ranges of the received signals. The second dimension represents multiple spatial channels such as different sensors or beams. The third dimension, slow time, represent pulses. Range filtering operates along the fast-time dimension of the cube. Processing along the other dimensions is not performed. If the data contains only one channel or pulse, the data cube can contain fewer than three dimensions. Because this object performs no Doppler processing, you can use it to process noncoherent radar pulses.

The output of the block is also a data cube with the same number of dimensions as the input. Its first dimension contains range-processed data but its length can differ from the first dimension of the input data cube.

Ports

Input

X — Input data cube
complex-valued *K*-by-1 column vector | complex-valued *K*-by-*L* matrix | complex-valued *K*-by-*N*-by-*L* array

Input data cube, specified as a complex-valued *K*-by-1 column vector, a complex-valued *K*-by-*L* matrix, or a complex-valued *K*-by-*N*-by-*L* array.

- *K* is the number of range or time samples.
- *N* is the number of independent channels such as sensors or directions.
- *L* is the number of pulses or sweeps in the input signal.

See “Radar Data Cube Concept”.

Each *K*-element column vector is processed independently.

For an FMCW waveform, with a triangle sweep, the sweeps alternate between positive and negative slopes. However, Range Response is designed to process consecutive sweeps of the same slope. To apply the Range Response block for a triangle-sweep system, use one of the following approaches:

- Specify a positive **Sweep slope** parameter value, with X corresponding to upsweeps only. After obtaining the Doppler or speed values, divide them by 2.
- Specify a negative **Sweep slope** parameter value, with X corresponding to downsweeps only. After obtaining the Doppler or speed values, divide them by 2.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Coeff — Matched filter coefficients
complex-valued column vector

Matched filter coefficients, specified as a complex-valued column vector. The length of the vector must be less than or equal to the number of rows in the input data, *K*.

Dependencies

To enable this port, set **Range processing method** to `Matched filter`.

Data Types: `double`

XRef — Reference signal
complex-valued *K*-by-1 column vector

Reference signal used for dechirping the input signal, specified as a complex-valued *K*-by-1 column vector. The number of rows must equal the length of the first dimension of X.

Dependencies

To enable this port, set **Range processing method** to `FFT` and select the **Dechirp input signal** parameter.

Data Types: `double`

Output

Resp — Range response data cube
complex-valued *M*-element column vector | complex-valued *M*-by-*L* matrix | complex-valued *M*-by-*N* by-*L* array

Range response data cube, returned as a

- Complex-valued *M*-element column vector
- Complex-valued *M*-by-*L* matrix
- Complex-valued *M*-by-*N* by-*L* array

See “Radar Data Cube Concept”. The value of *M* depends on the type of processing

Range processing method	Dechirp input signal	Value of *M*
`FFT`	`off`	If you set the **Source of FFT length in range processing** to `Auto`,$M = K$, the length of the first dimension of `x`. Otherwise, *M* equals the value of the **FFT length in range processing** parameter.
	`on`	*M* equals the number of rows, *K*, of the input signal.
`Matched filter`	N/A	*M* equals the number of rows, *K*, of the input signal.

Data Types: `double`

Range — Range values along range dimension
real-valued *M*-by-1 column vector

Range values along the first dimension of the **Resp** output data port, specified as a real-valued *M*-by-1 column vector. This quantity defines the range values along the first dimension of the `Resp` output port data. Units are in meters.

Data Types: `double`

Parameters

Range processing method — Range processing method
`Matched filter` (default) | `FFT`

Range processing method, specified as `Matched filter` or `FFT`.

`Matched filter`	The block applies a matched filter to the incoming signal. This approach is commonly used for pulsed signals, where the matched filter is the time reverse of the transmitted signal.
`FFT`	The block applies an FFT to the input signal. This approach is commonly used for FMCW and linear FM pulsed signals.

Data Types: `char`

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`.

Data Types: `double`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

FM sweep slope (Hz/s) — FM sweep slope
`1e9` (default) | scalar

Specify the slope of the linear FM sweep as a scalar. This parameter must match the actual sweep of the input data in port X.

Dependencies

To enable this parameter, set **Range processing method** to `FFT`.

Data Types: `double`

Dechirp input signal — Enable dechirping of input signal
on (default) | `off`

Select this parameter to enable dechirping of input signal.

Dependencies

To enable this parameter, set **Range processing method** to `FFT`.

Data Types: `Boolean`

Source of FFT length in range processing — Source of FFT length for range processing
`Auto` (default) | `Property`

Source of FFT length for range processing, specified as `Auto` or `Property`

`Auto`	The FFT length equals the number of rows of the input data cube.
`Property`	Specify FFT length in the **FFT length in range processing** parameter.

Dependencies

To enable this parameter, set **Range processing method** to `FFT`.

Data Types: `char`

FFT length in range processing — Range processing FFT length
`1024` (default) | positive integer

FFT length for range processing, specified as a positive integer.

Dependencies

To enable this parameter, set **Range processing method** to `FFT` and **Source of FFT length in range processing** to `Property`.

Data Types: `double`

Range processing window — Range FFT weighting window
`None` (default) | `Hamming` | `Chebyshev` | `Hann` | `Kaiser` | `Taylor`

Range FFT weighting window, specified as `None`, `Hamming`, `Chebyshev`, `Hann`, `Kaiser`, or `Taylor`.

If you set this property to `Taylor`, the generated Taylor window has four nearly constant sidelobes next to the mainlobe.

Dependencies

To enable this parameter, set **Range processing method** to `FFT`.

Data Types: `char`

Range sidelobe attenuation level — Sidelobe attenuation for range processing
`30` (default) | positive scalar

Sidelobe attenuation for range processing, specified as a positive scalar. Units are in dB.

Dependencies

To enable this parameter, set **Range processing method** to `FFT` and **Range processing window** to `Kaiser`, `Chebyshev`, or `Taylor`.

Data Types: `double`

Set reference range at center — Set reference range at center of range grid
`on` (default) | `off`

Set reference range at the center of range grid, specified as `on` or `off`. Selecting this check box, enables you to set the reference range at the center of the range grid. Otherwise, the reference range is set to the beginning of the range grid.

Dependencies

To enable this property, set the **Range processing method** to `FFT`.

Reference range (m) — Reference range of range grid
`0.0` (default) | nonnegative scalar

Reference range of the range grid, specified as a nonnegative scalar.

- If you set the **Range processing method** parameter to `Matched filter`, the reference range is set to the start of the range grid.
- If you set the **Range processing method** property to `FFT`, the reference range depends on the **Set reference range at center** check box.
  - When you select the **Set reference range at center** check box, the reference range is set to the center of the range grid.
  - If you do not select the **Set reference range at center** check box, the reference range is set to the start of the range grid.

  Units are in meters.

Example: `1000.0`

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`

Default: `Interpreted Execution`

Version History

Introduced in R2017a

References

[1] Richards, M. *Fundamentals of Radar Signal Processing, 2nd ed*. McGraw-Hill Professional Engineering, 2014.

[2] Richards, M., J. Scheer, and W. Holm, *Principles of Modern Radar: Basic Principles*. SciTech Publishing, 2010.

See Also

Blocks
Range Doppler Response

Functions
`chebwin` | `dechirp` | `hann` | `hamming` | `kaiser` | `taylorwin`

Objects
`phased.RangeResponse` | `phased.RangeDopplerResponse` | `phased.CFARDetector` | `phased.CFARDetector2D`

Receiver Preamp

Receiver preamplifier

Libraries:
Phased Array System Toolbox / Transmitters and Receivers

Description

The Receiver Preamp block implements a receiver preamplifier that amplifies an input signal and adds thermal noise. In addition, you can add phase noise using an input port.

Ports

Input

X — Input signal
complex-valued matrix

Input signal, specified as a complex-valued matrix.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

TR — Enable enabling signal
logical-valued scalar | scalar

Enable enabling signal, specified as a column vector whose length equals the number of rows in the input signal X. Every element of TR that equals `0` or `false` indicates that the receiver is turned off, and no input signal passes through the receiver. Every element of TR that is nonzero or `true` indicates that the receiver is turned on, and the input signal passes through.

Dependencies

To enable this port, select the **Enable enabling signal input** check box.

Data Types: `double` | `Boolean`

Ph — Input phase noise
scalar

Phase noise for each input sample of X, specified as a column vector whose length equals the number of rows in X.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this port, select the **Enable phase noise input** check box.

Data Types: `double`

Output

Out 1 — Output
complex-valued matrix

Output signal, returned as a complex-valued matrix.

Data Types: `double`

Parameters

Gain (dB) — Gain of receiver

`20` (default)

Specify a scalar containing the gain in dB of the receiver preamplifier.

Loss factor (dB) — Loss factor of receiver

`0` (default)

Specify a scalar containing the loss factor in dB of the receiver preamplifier.

Noise specification method — Noise specification method

`Noise temperature` (default) | `Noise power`

Specify the receiver noise as `Noise power` or `Noise temperature`.

Noise power (W) — Noise power

`0` (default)

Specify a scalar containing the noise power in watts at the receiver preamplifier. If the receiver has multiple channels or sensors, the noise bandwidth applies to each channel or sensor.

Dependencies

This parameter appears only when you set **Noise specification method** to `Noise power`.

Noise figure (dB) — Noise figure of receiver

`0` (default)

Specify a scalar containing the noise figure of the receiver preamplifier. Units are in dB. If the receiver has multiple channels or sensors, the noise figure applies to each channel or sensor. This parameter appears only when you set **Noise specification method** to `Noise temperature`.

Dependencies

This parameter appears only when you set **Noise specification method** to `Noise temperature`.

Reference temperature (K) — Reference temperature of receiver

290 (default)

A scalar containing the reference temperature in degrees kelvin of the receiver preamplifier. If the receiver has multiple channels or sensors, the reference temperature applies to each channel or sensor.

Dependencies

This parameter appears only when you set **Noise specification method** to `Noise temperature`.

Inherit sample rate — Inherit sample rate

on (default) | off

Select this check box to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate** parameter.

Dependencies

This parameter appears only when you set **Noise specification method** to `Noise temperature`.

Sample rate (Hz) — Sampling rate of signal
1e6 (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Enable enabling signal input — Add input to specify enabling signal

off (default) | on

Select this check box to allow input of the receiver-enabling signal via the input port TR. T

Dependencies

This parameter appears only when you set **Noise specification method** to `Noise temperature`.

Enable phase noise input — Add input to specify phase noise

off (default) | on

Select this check box to allow input of phase noise for each incoming sample using the input port Ph. You can use this information to emulate coherent-on-receive systems.

Dependencies

This parameter appears only when you set **Noise specification method** to `Noise temperature`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><td rowspan="2">Block Simulation</td><td colspan="3">Simulation Behavior</td></tr>
<tr><td><code>Normal</code></td><td><code>Accelerator</code></td><td><code>Rapid Accelerator</code></td></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

Introduced in R2014b

See Also

`phased.ReceiverPreamp`

Rectangular Waveform

Rectangular pulse waveform

Libraries:
Phased Array System Toolbox / Detection

Description

The Rectangular Waveform block generates a rectangular pulse waveform with a specified pulse width and pulse repetition frequency (PRF). The block outputs an integral number of pulses or samples.

Ports

Input

PRFIdx — PRF Index
positive integer

Index to select the pulse repetition frequency (PRF), specified as a positive integer. The index selects the PRF from the predefined vector of values specified by the **Pulse repetition frequency (Hz)** parameter.

Example: 4

Dependencies

To enable this port, select **Enable PRF selection input**.

Data Types: `double`

FreqOffset — Frequency offset
scalar

Frequency offset in Hz, specified as a scalar.

Example: `2e3`

Dependencies

To enable this port, set **Source of Frequency Offset** to `Input port`.

Data Types: `double`

Output

Y — Pulse waveform
complex-valued vector

Pulse waveform samples, returned as a complex-valued vector.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, returned as a positive scalar.

Dependencies

To enable this port, set the **Output signal format** parameter to `Pulses` and then select the **Enable PRF output** parameter.

Data Types: `double`

Coeff — Matched filter coefficients
vector | matrix

Matched filter coefficients, returned as a vector or matrix.

Dependencies

To enable this port, select **Enable Matched Filter Coefficients Output**.

Data Types: `double`

Parameters

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.

Programmatic Use
Block Parameter:`SampleRate`
Type:double
Values:positive scalar
Default:`1e6`

Method to specify pulse duration — Pulse duration as time or duty cycle
`Pulse width` (default) | `Duty cycle`

Method to set the pulse duration, specified as `Pulse width` or `Duty cycle`. When you set this parameter to `Pulse width`, the pulse duration is set using the **Pulse width (s)** parameter. When you set this parameter to `Duty cycle`, the pulse duration is computed from the values of the **Pulse repetition frequency (Hz)** and **Duty Cycle** parameters.

Programmatic Use
Block Parameter:`DurationSpecification`
Type:string
Values:string
Default:`'Pulse width'`

Pulse width (s) — Time duration of pulse
`50e-6` (default) | positive scalar

The duration of each pulse, specified as a positive scalar. Set the product of **Pulse width (s)** and **Pulse repetition frequency** to be less than or equal to one. This restriction ensures that the pulse width is smaller than the pulse repetition interval. Units are in seconds.

Example: `300e-6`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Pulse width`.

Programmatic Use

Block Parameter:`PulseWidth`
Type:double
Values:string
Default:`50e-6`

Duty cycle — Waveform duty cycle
`0.5` (default) | scalar in the range *[0,1]*

Waveform duty cycle, specified as a scalar in the range *[0,1]*.

Example: `0.7`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Duty cycle`.

Programmatic Use

Block Parameter:`DutyCycle`
Type:double
Values:positive scalar
Default:`1e6`

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1e4` (default) | positive scalar

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The value of **Pulse repetition frequency (Hz)** must satisfy these constraints:

- The product of **Pulse width** and **Pulse repetition frequency (Hz)** must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of **Pulse repetition frequency** must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* by using block parameter settings alone or in conjunction with the input port, `PRFIdx`.

- When the **Enable PRF selection input** parameter is not selected, set the *PRF* using block parameters.

  - To implement a constant *PRF*, specify **Pulse repetition frequency (Hz)** as a positive scalar.
  - To implement a staggered *PRF*, specify **Pulse repetition frequency (Hz)** as a row vector with positive values. After the waveform reaches the last element of the vector, the process continues cyclically with the first element of the vector. When *PRF* is staggered, the time between successive output pulses cycles through the successive values of the *PRF* vector.
- When the **Enable PRF selection input** parameter is selected, you can implement a selectable *PRF* by specifying **Pulse repetition frequency (Hz)** as a row vector with positive real-valued entries. But this time, when you execute the block, select a *PRF* by passing an index into the *PRF* vector into the `PRFIdx` port.

In all cases, the number of output samples is fixed when you set the **Output signal format** to `Samples`. When you use a varying *PRF* and set **Output signal format** to `Pulses`, the number of output samples can vary.

Programmatic Use
Block Parameter:PRF
Type:double
Values:positive scalar
Default:`1e6`

Enable PRF selection input — Select predefined PRF
off (default) | on

Select this parameter to enable the `PRFIdx` port.

- When enabled, pass in an index into a vector of predefined PRFs. Set predefined PRFs using the **Pulse repetition frequency (Hz)** parameter.
- When not enabled, the block cycles through the vector of PRFs specified by the **Pulse repetition frequency (Hz)** parameter. If **Pulse repetition frequency (Hz)** is a scalar, the PRF is constant.

Programmatic Use
Block Parameter:`PRFSelectionInputPort`
Type:logical
Values:positive scalar
Default:`off`

Source of Frequency Offset — Source of frequency offset
`Property` (default) | `Input port`

Source of frequency offset, specified as `Property` or `Input port`.

- When set to `Property`, the offset is determined by the value of the **Frequency Offset** parameter.
- When set to `Input port`, the offset is determined by the value of the **FreqOffset** port.

Programmatic Use
Block Parameter:`FrequencyOffsetSource`
Type:enum
Values:`Property`, `Input Port`
Default:`Property`

Frequency Offset (Hz) — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: 2e3

Dependencies

To enable this parameter set the **Source of Frequency Offset** parameter to `Input port`.

Programmatic Use
Block Parameter:`FrequencyOffset`
Type:double
Values:`scalar`
Default:`0`

Source of simulation sample time — Source of simulation sample time
`Derive from waveform parameters` (default) | `Inherit from Simulink engine`

Source of simulation sample time, specified as `Derive from waveform parameters` or `Inherit from Simulink engine`. When set to `Derive from waveform parameters`, the block runs at a variable rate determined by the PRF of the selected waveform. The elapsed time is variable. When set to `Inherit from Simulink engine`, the block runs at a fixed rate so the elapsed time is a constant.

Dependencies

To enable this parameter, select the **Enable PRF selection input** parameter.

Programmatic Use
Block Parameter:`SimulationTimeSource`
Type:enum
Values:`Derive from waveform parameters`, `Inherit from Simulink engine`
Default:`Derive from waveform parameters`

Output signal format — Format of the output signal
`Pulses` (default) | `Samples`

The format of the output signal, specified as `Pulses` or `Samples`.

If you set this parameter to `Samples`, the output of the block consists of multiple samples. The number of samples is the value of the **Number of samples in output** parameter.

If you set this parameter to `Pulses`, the output of the block consists of multiple pulses. The number of pulses is the value of the **Number of pulses in output** parameter.

Programmatic Use
Block Parameter:`OutputFormat`
Type:enum
Values:`Pulses` `Samples`
Default:`Pulses`

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use

Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Number of pulses in output — Number of pulses in output
`1` (default) | positive integer

Number of pulses in the block output, specified as a positive integer.

Example: `2`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Pulses`.

Programmatic Use

Block Parameter:`NumPulses`
Type:double
Values:`positive scalar`
Default:`1`

Data Types: `double`

Enable PRF Output — Enable output of PRF
`off` (default) | `on`

Select this parameter to enable the PRF output port.

Dependencies

To enable this parameter, set **Output signal format** to `Pulses`.

Programmatic Use

Block Parameter:`PRFOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Enable Matched Filter Coeficients Output — Enable output of matched filter coefficients
`off` (default) | `on`

Select this parameter to enable the `Coeff` output port.

Programmatic Use
Block Parameter:`CoefficientOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

Introduced in R2014b

See Also

`phased.RectangularWaveform`

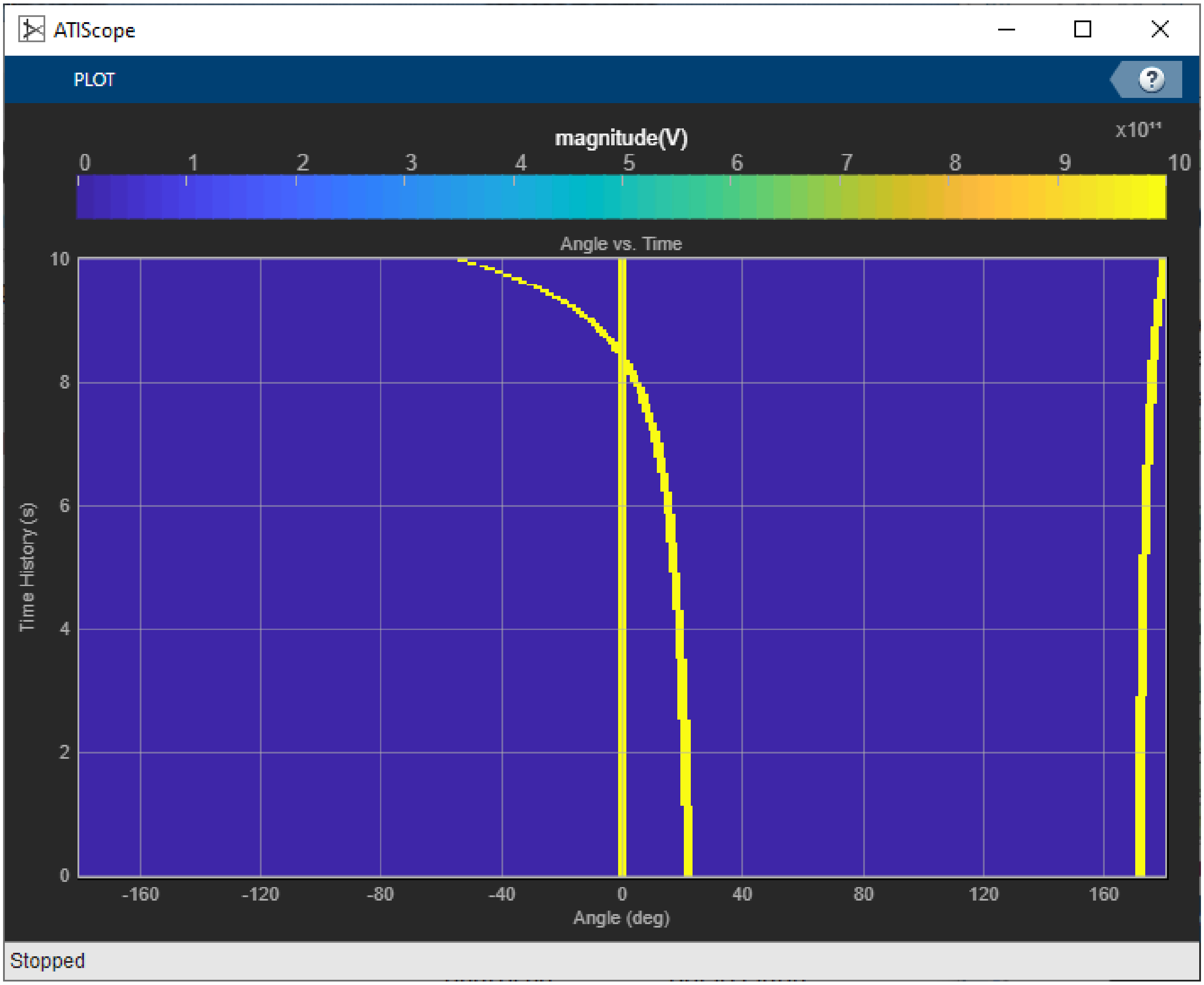
Angle-Time Intensity Scope

Angle-time intensity scope

Libraries:
Phased Array System Toolbox / Sinks

Description

The Angle-Time Intensity Scope block creates a scrolling display of angle response intensities as a function of time. The input consists of angle responses for a pulse or FMCW signal. Each frame of data creates a new line on the scope. The scope serves as a display of the angle response.



Ports

Input

X — Input data
real-valued length-*M* vector

Input data, specified as a real-valued length-*M* vector of angle responses. Each vector corresponds to a line of data. The block can accept single precision data type input, but converts the data to double precision for display.

Data Types: `double` | `single`

Parameters

Angle offset (degree) — Angle offset
`0.0` (default) | scalar

Angle offset, specified as a scalar. This property defines the angle value of the first column of the display. Units are in degrees.

Programmatic Use
Block Parameter:`AngleOffset`
Type:double
Values:`scalar |`
Default:0.0

Data Types: `double`

Angle resolution (degree) — Angle difference between samples
`1.0` (default) | positive scalar

Angle separation between samples, specified as a positive scalar. This property defines the angle difference between columns of the scope. Units are in degrees.

Programmatic Use
Block Parameter:`AngleResolution`
Type:double
Values:`positive scalar |`
Default:0.0

Data Types: `double`

Time span (s) — Time span of display
`Auto` (default)

Time span of the intensity display, specified as a positive scalar. When the **Time span (s)** is set to `Auto` the simulation stops at the default **stop time** of the simulation. Units are in seconds.

Programmatic Use
Block Parameter:`TimeSpan`
Type:double
Values:scalar
Default:`0.1`

Data Types: `double`

Time resolution (s) — Time difference between rows
`Auto` (default)

Time interval between rows, specified as a positive scalar. This property defines the time duration between rows of the scope. When the **Time resolution (s)** parameter is set to `Auto` the block uses the **Compiled Sample Time** value during simulation. At all other times, the block uses the value specified by this parameter. Units are in seconds.

Programmatic Use
Block Parameter:`TimeResolution`
Type:double
Values:positive scalar

Default:0.001

Data Types: double

Grid — Show Cartesian grid overlay
on (default) | off

Show Cartesian grid overlay. To show the grid, select the check box.

Programmatic Use
Block Parameter: ShowGrid
Type:logical
Values:1 | 0
Default:1

Ticks — Show tick value labels
on (default) | off

Show value labels on tick marks. To display the labels, select the check box.

Programmatic Use
Block Parameter: ShowTicks
Type:logical
Values:1 | 0
Default:1

Labels

Angle label — Angle-axis label
'Angle (degree)' (default) | character vector | string

Angle-axis label, specified as a character vector or a string.

Example: 'Angle (rad)'

Tunable: Yes

Programmatic Use
Block Parameter: AngleLabel
Type:string
Values:string
Default:'Angle (degree)'

Data Types: char | string

Time label — Time axis label
'Time History (s)' (default) | string

Time-axis label, specified as a string.

Tunable: Yes

Programmatic Use
Block Parameter: TimeLabel
Type:string
Values:string
Default:'Time History (s)'

Data Types: `string`

Title — Window title
`Angle vs. Time'` (default) | string

Window title, specified as a string.

Tunable: Yes

Programmatic Use
Block Parameter:`Title`
Type:string
Values:string
Default:`'Doppler vs. Time'`

Data Types: `char`

Colorbar label — Color bar label
`'dB'` (default) | string

Color bar label, specified as a string.

Tunable: Yes

Programmatic Use
Block Parameter:`ColorBarLabel`
Type:string
Values:string
Default:`'dB'`

Data Types: `string`

Block Characteristics

Data Types	`double` | `single`
Direct Feedthrough	`no`
Multidimensional Signals	`no`
Variable-Size Signals	`no`
Zero-Crossing Detection	`no`

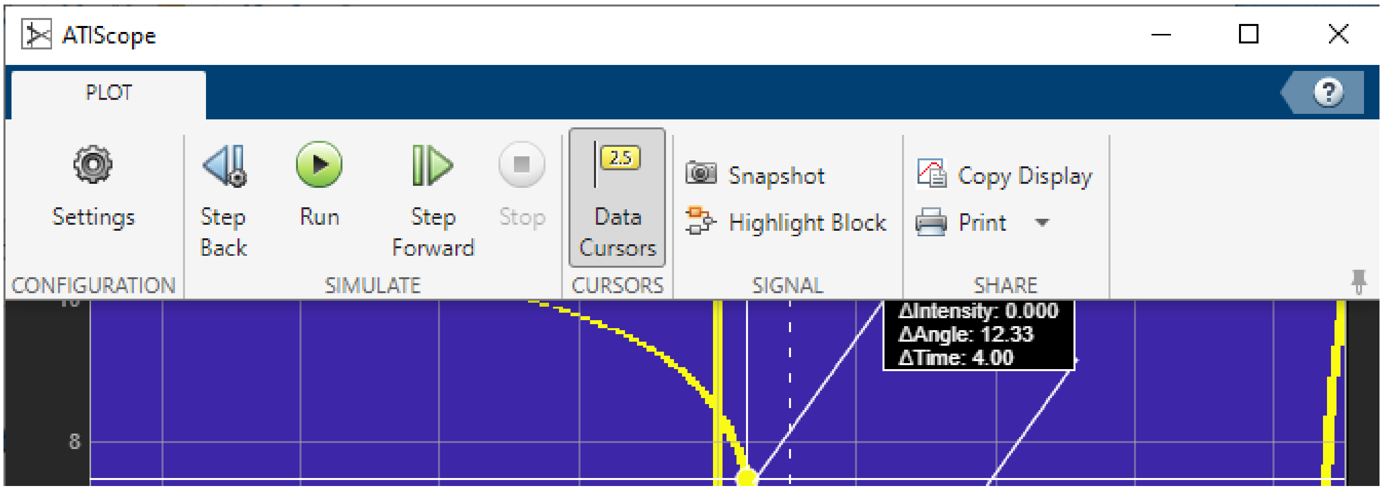
More About

Description

This section shows how to run the scope, configure the settings to show labels for the various features and use the cursor to examine data. The scope is based on the `phased.ATIScope` System object. Using the scope to display response corresponds to setting the `IQDataInput` property of the `phased.ATIScope` System object to `false`.

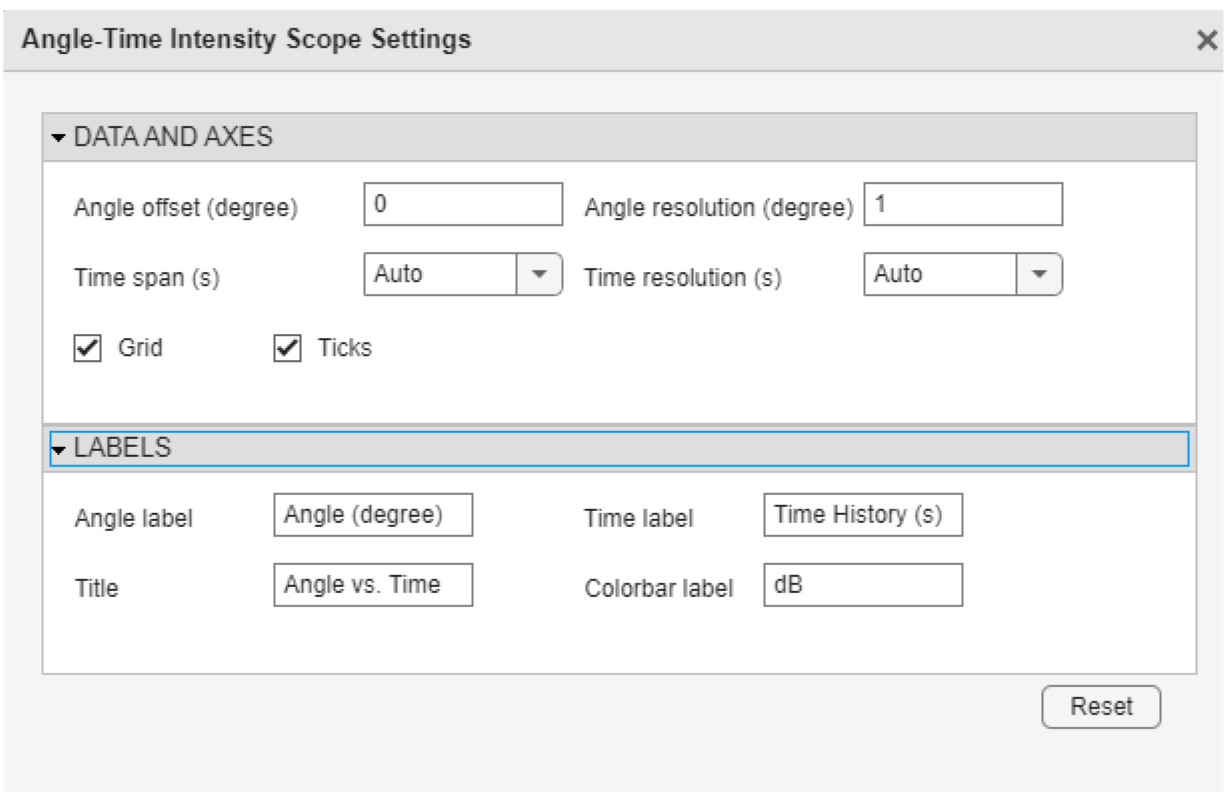
Run the Scope

After selecting and opening the block, the scope displays a single tab **PLOT**. Click the tab to show all the controls available for the block. The **Settings** menu displays all the settable parameters for the scope. The **Step Back**, **Run**, **Step Forward**, and **Stop** buttons control the simulation. The **Data Cursors** button lets you examine data values on the display.



Scope Parameter Settings

Clicking the **Settings** button on the **PLOT** tab opens the parameter window with the **DATA AND AXES** and **LABELS** parameters.

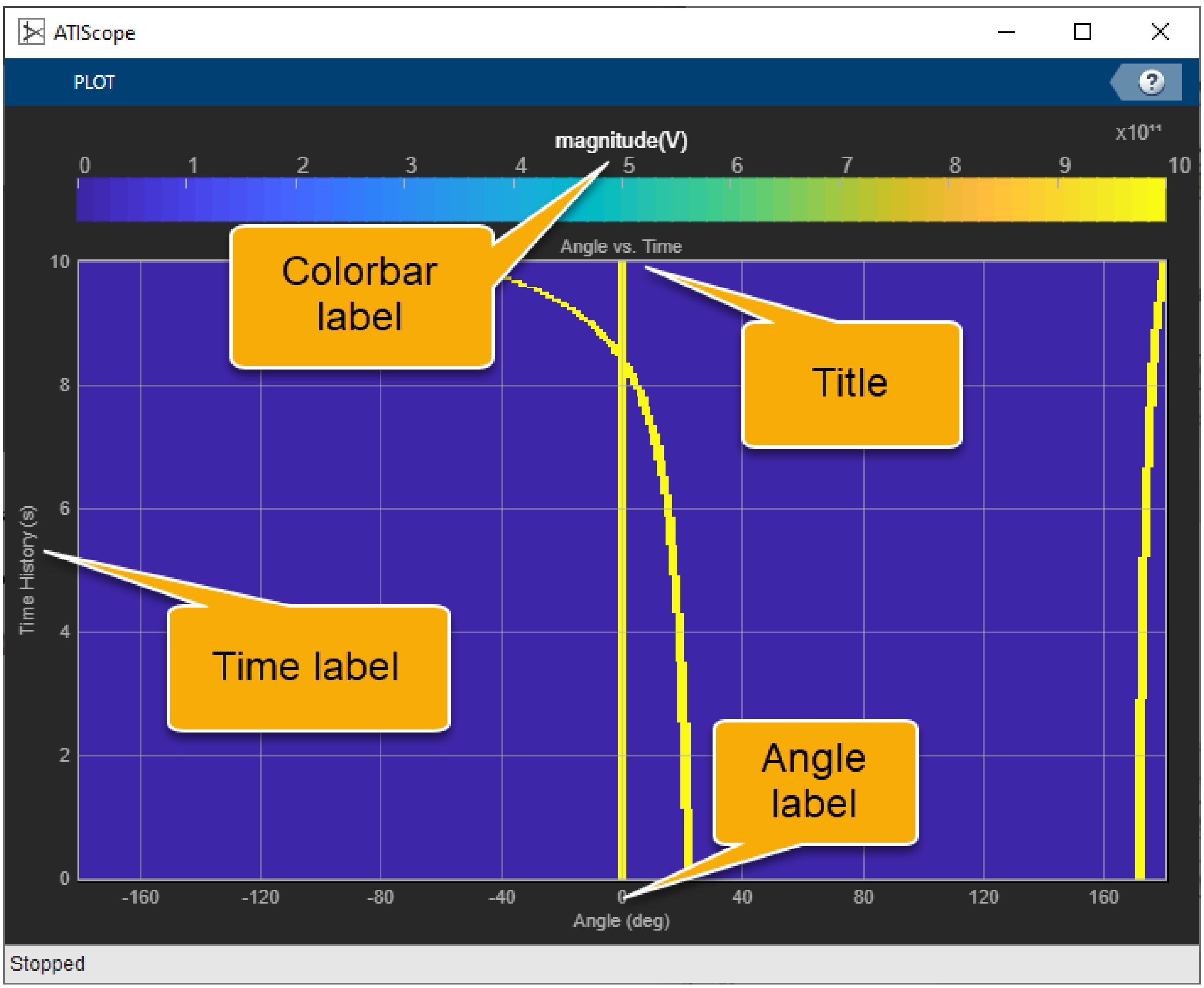


DATA AND AXES

- **Angle offset (Hz)** – Use this field to set the frequency offset of the *x*-axis.
- **Angle resolution (Hz)** – Use this field to set the granularity of the x-axis.
- **Time span (s)** – Use this field to set the duration of the y-axis.
- **Time resolution (s)** – Use this field to set the granularity of the y-axis.

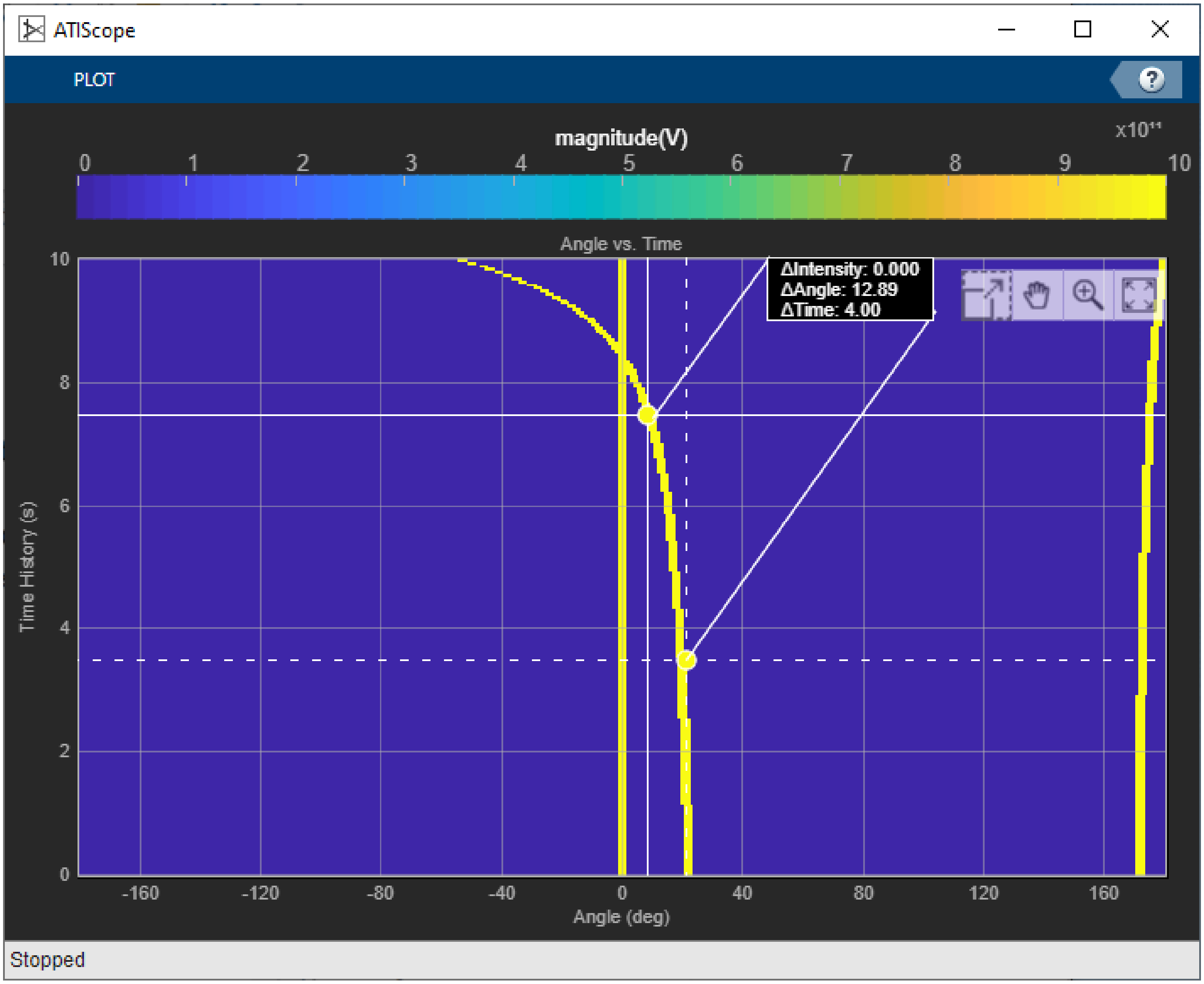
LABELS

- **Angle label** – Use this field to set the *y*-axis label.
- **Time label** – Use this field to set the *y*-axis label.
- **Title** – Use this field to set the display title.
- **Colorbar label** – Set this field to set the color bar label.



Using Cursors

Click the **Data Cursors** button to display the screen cursors. Each cursor consists of intersecting horizontal and vertical cursors that define a point on the display. Positioning two cursors on the display shows the difference between the intensities at the two cursor points.



Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

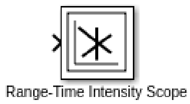
This block can be used for simulation visibility in systems that generate code but is not included in the generated code.

See Also

Range-Time Intensity Scope | Doppler-Time Intensity Scope | phased.ATIScope | `phased.DTIScope` | `phased.RTIScope`

Range-Time Intensity Scope

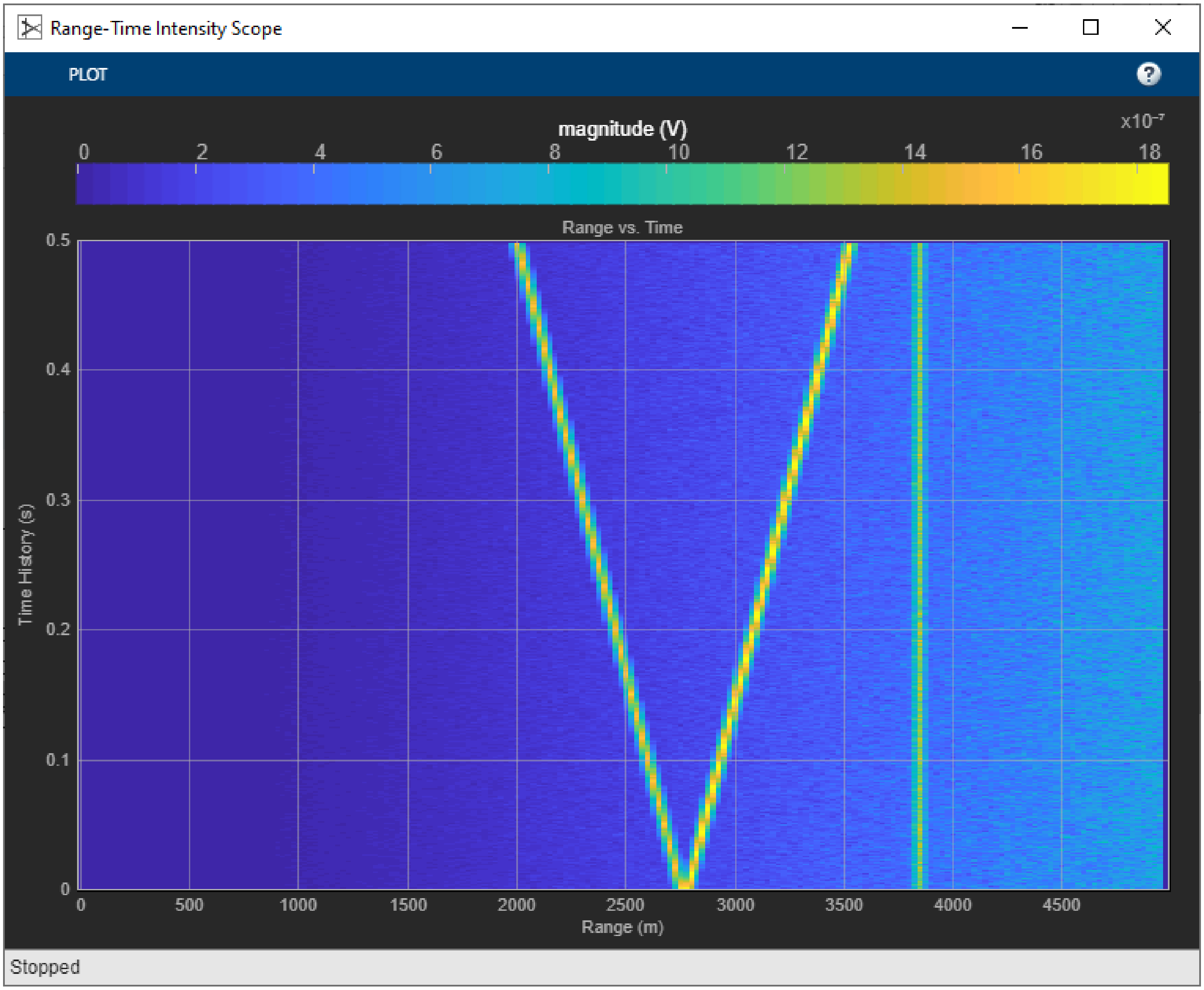
Range-time intensity scope



Libraries:
Phased Array System Toolbox / Sinks

Description

The Range-Tine Intensity Scope block creates a scrolling display of range response intensities as a function of time. The input consists of range responses for a pulse or FMCW signal. Each frame of data creates a new line on the scope. The scope serves only as a display of the range response. Using the response as input corresponds to setting the `IQDataInput` property of the `phased.RTIScope` System object to `false`.



Ports

Input

X — Input data
real-valued length-M vector

Input data, specified as a real-valued length-M vector of range responses. Each vector corresponds to a line of data. The block accepts single precision data type input, but converts the data to double precision for display.

Data Types: `double` | `single`

Parameters

Data and Axes

Range Offset (m) — Range offset
`0.0` (default) | positive scalar

Range offset, specified as a positive scalar. This parameter defines the range value of the first column of the display. Units are in meters.

Programmatic Use
Block Parameter:`RangeOffset`
Type:double
Values:scalar
Default:`0`

Data Types: `double`

Time Span (s) — Time span of display
`0.1` (default) | positive scalar

Time span of the intensity display, specified as a positive scalar. Units are in seconds.

Programmatic Use
Block Parameter:`TimeSpan`
Type:double
Values:scalar
Default:`1`

Data Types: `double`

Range Resolution (m) — Range difference between samples
`1` (default) | positive scalar

Range distance between samples, specified as a positive scalar. This parameter defines the distance between columns of the scope. Units are in meters.

Programmatic Use
Block Parameter:`RangeResolution`
Type:double
Values:positive scalar
Default:`25`

Data Types: `double`

Time Resolution (s) — Time difference between rows
`0.001` (default) | positive scalar

Time interval between samples, specified as a positive scalar. This parameter defines the time interval between the rows of the scope. Units are in seconds.

Programmatic Use
Block Parameter:`TimeResolution`
Type:double
Values:positive scalar
Default:`0.00033356`

Data Types: `double`

Show Grid — Show Cartesian grid overlay
on (default) | off

Show Cartesian grid overlay. To show the grid, select the check box.

Programmatic Use
Block Parameter:`ShowGrid`
Type:logical
Values:1 | 0
Default:`1`

Show Ticks — Show tick value labels
on (default) | off

Show value labels on tick marks. The display the labels on, select the check box.

Programmatic Use
Block Parameter:`ShowTicks`
Type:logical
Values:1 | 0
Default:`1`

Labels

Range Label — Range axis label
`'Range (m)'` (default) | string

Range-axis label, specified as a string.

Tunable: Yes

Programmatic Use
Block Parameter:`RangeLabel`
Type:string
Values:string
Default:`'Range (m)'`

Data Types: `char`

Time Label — Time axis label
`Time History (s)'` (default) | string

Time-axis label, specified as a string.

Tunable: Yes

Programmatic Use
Block Parameter:`TimeLabel`
Type:string
Values:string
Default:`'Time History (s)'`

Data Types: `string`

Title — Window title
`'Range vs. Time'` (default) | string

Window title, specified as a string.

Tunable: Yes

Programmatic Use
Block Parameter:`Title`
Type:string
Values:string
Default:`Range vs. Time'`

Data Types: `char`

Colorbar Label — Color bar
`'mag'` (default) | string

Color bar label, specified as a string.

Tunable: Yes

Programmatic Use
Block Parameter:`ColorBarLabel`
Type:string
Values:string
Default:`'mag'`

Data Types: `string`

Block Characteristics

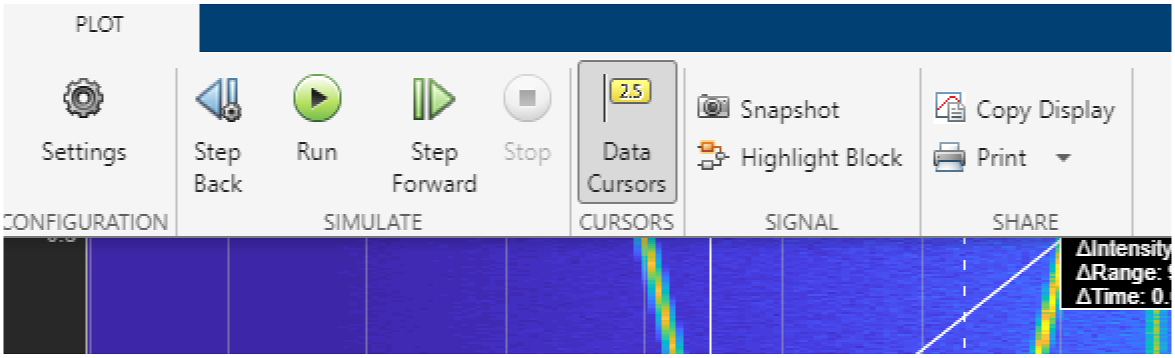
Data Types	`double` | `single`
Direct Feedthrough	`no`
Multidimensional Signals	`no`
Variable-Size Signals	`no`
Zero-Crossing Detection	`no`

Algorithms

This section shows how to run the scope, configure the settings to show labels for the various features and use the cursor to examine data.

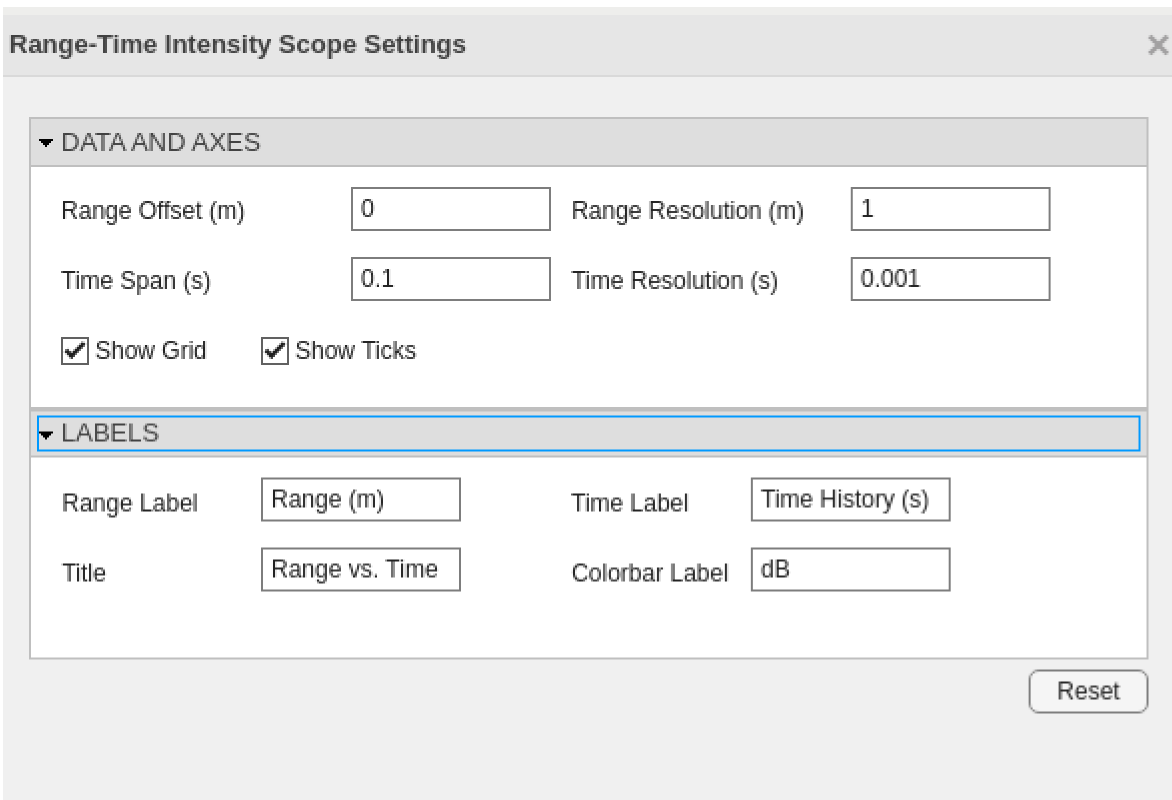
Run the Scope

After selecting and opening the block, the scope displays a single tab **PLOT**. Opening this tab shows all the controls available for the block. The **Settings** menu displays all the settable parameters for the scope. The **Step Back**, **Run**, **Step Forward**, and **Stop** buttons control the simulation. The **Data Cursors** button lets you examine data values on the display.



Scope Parameter Settings

Selecting the **Settings** menu from **PLOT** panel scope opens the parameter window with the **DATA AND AXES** and **LABELS** panels.

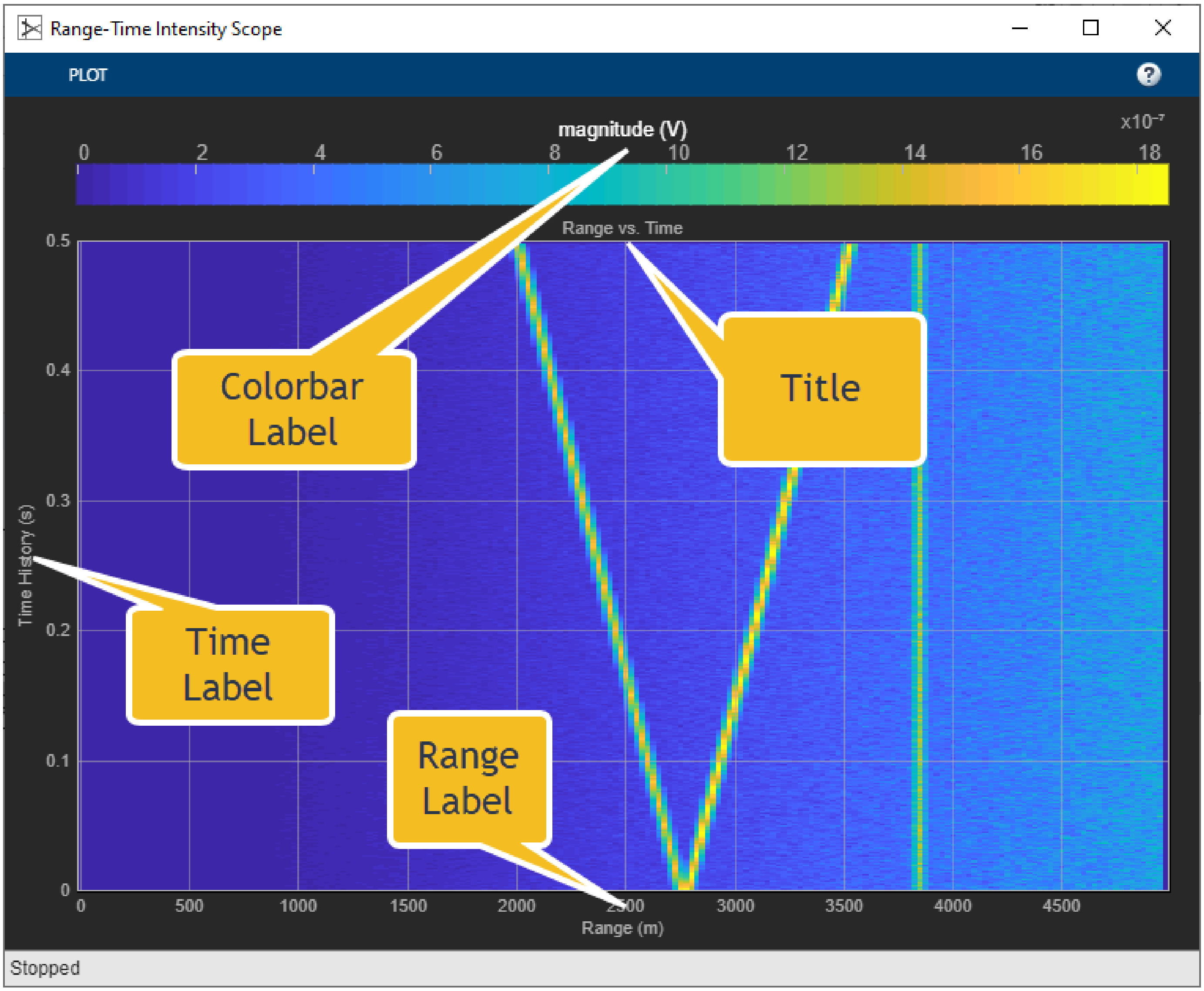


DATA AND AXES

- **Range Offset (m)** - Use this field to set the offset of the x-axis.
- **Range Resolution (m)** - Use this field to set the granularity of the x-axis.
- **Time Span (s)** - Use this field to set the duration of the y-axis.
- **Time Resolution (s)** - Use this field to set the granularity of the y-axis.

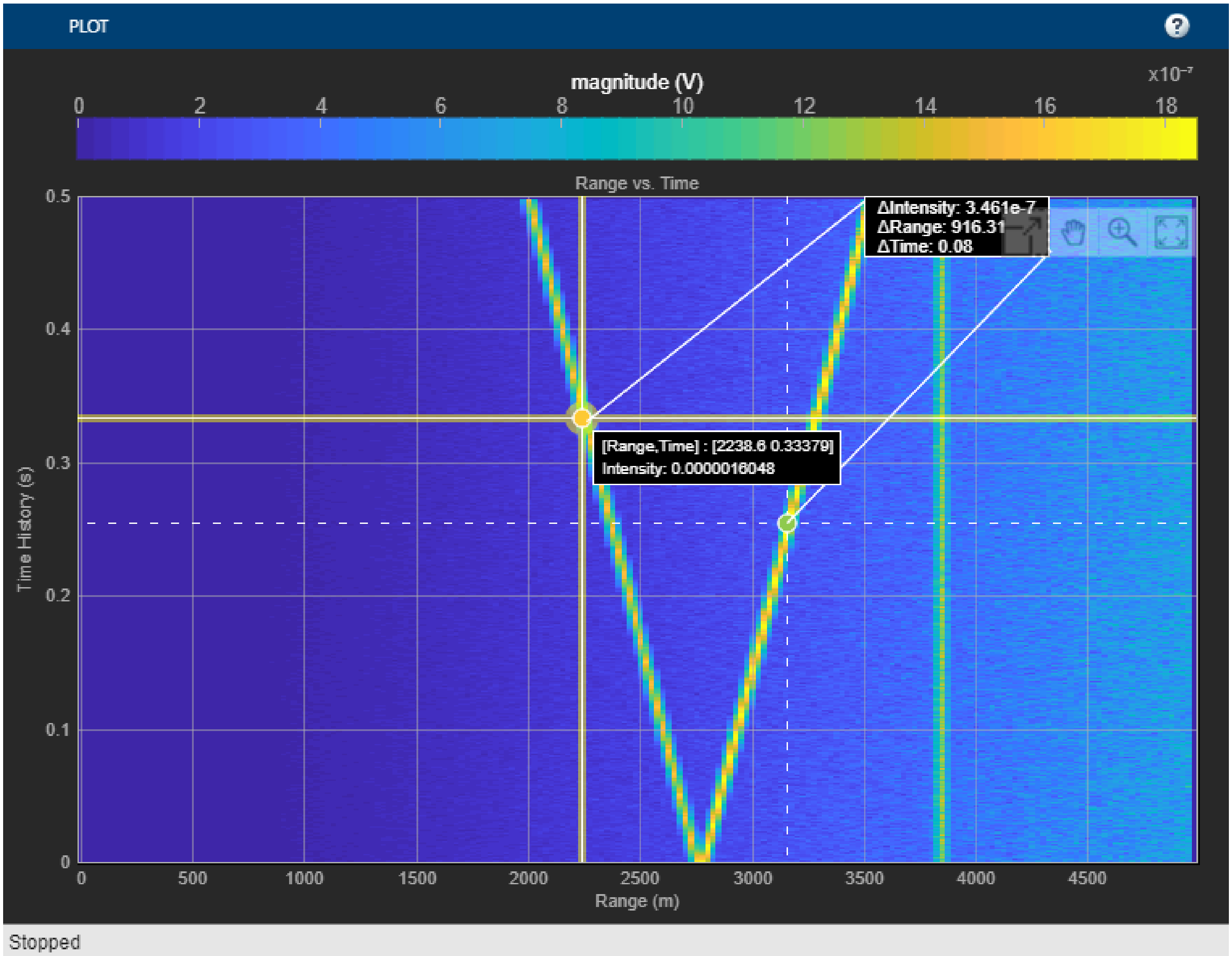
Labels

- **Range Label** – Use this field to set the x-axis label.
- **Time Label** – Use this field to set the y-axis label.
- **Title** – Use this field to set the display title.
- **Colorbar Label** – Use this field to set the color bar label.



Using Cursors

Select the **Cursors Button** to display the screen cursors. Each cursor consists of intersecting horizontal and vertical lines that define a point on the display. Positioning two cursors on the display shows the difference between the intensities at the two cursor points.



Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

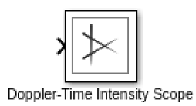
This block can be used for simulation visibility in systems that generate code but

See Also

Doppler-Time Intensity Scope | Angle-Time Intensity Scope | `phased.DTIScope` | `phased.RTIScope`

Doppler-Time Intensity Scope

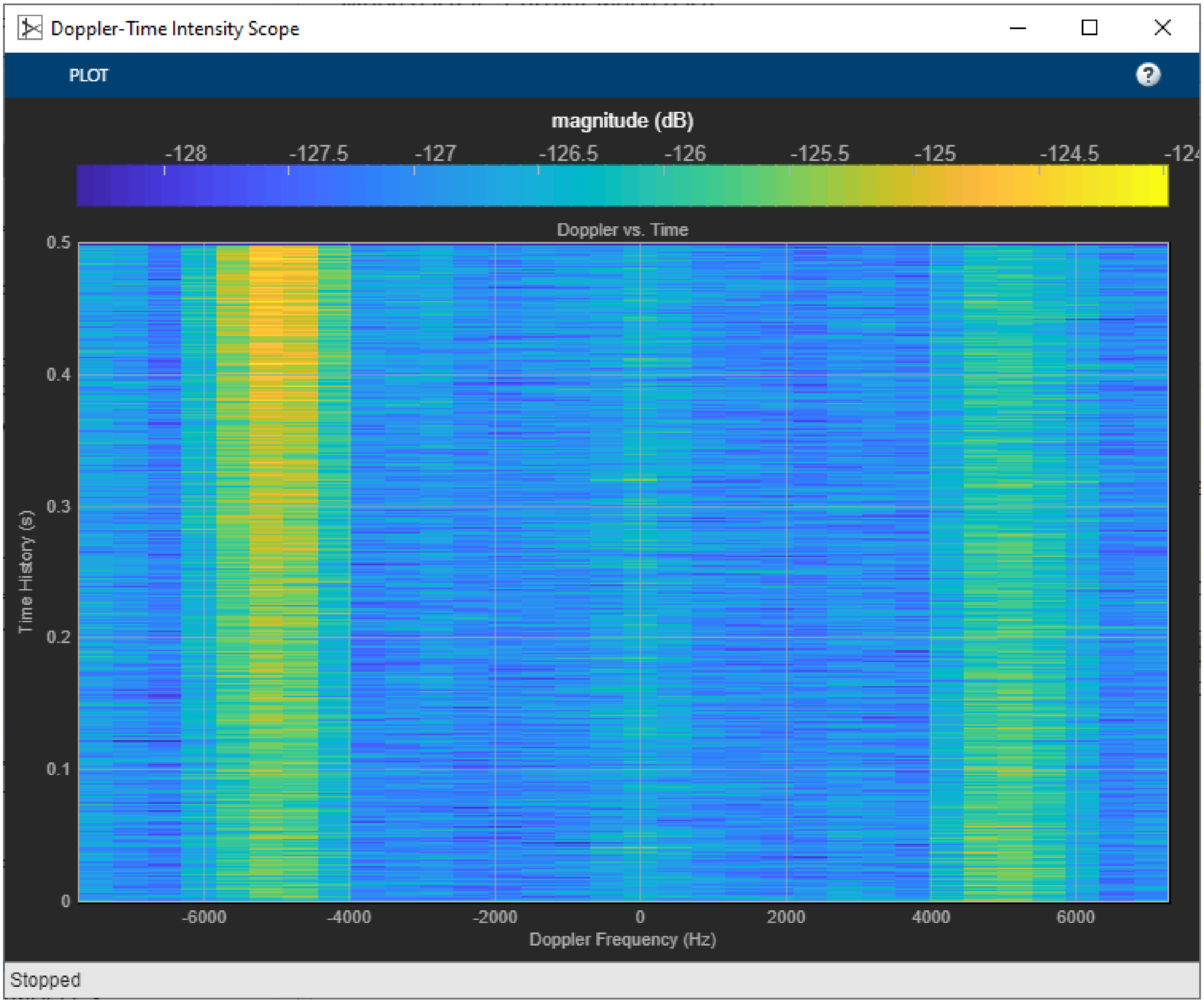
Doppler-time intensity scope



Libraries:
Phased Array System Toolbox / Sinks

Description

The Doppler-Time Intensity Scope block creates a scrolling display of Doppler response intensities as a function of time. The input consists of Doppler responses for a pulse or FMCW signal. Each frame of data creates a new line on the scope. The scope serves only as a display of the Doppler response. Using the response as input corresponds to setting the `IQDataInput` property of the `phased.DTIScope` System object to `false`.



Ports

Input

X — Input data
real-valued length-*M* vector

Input data, specified as a real-valued length-*M* vector of Doppler responses. Each vector corresponds to a line of data. The block accepts single precision data type input, but converts the data to double precision for display.

Data Types: `double` | `single`

Parameters

Doppler Output — Doppler output domain
'Frequency' (default) | 'Speed'

Doppler output domain, specified as 'Frequency' or 'Speed'. If you set this property to 'Frequency', the Doppler domain is Doppler shift. Units are in Hz. If you set this property to 'Speed'', the Doppler domain is radial speed. Units are in m/s.

Programmatic Use
Block Parameter: DopplerOutput
Type:character vector
Values:'Frequency' | 'Speed'
Default:'Frequency'

Data Types: string

Propagation Speed (m/s) — Signal propagation speed
physconst("LightSpeed") (default) | positive scalar

Signal propagation speed, specified as a positive scalar. To obtain the speed of light in SI units, use physconst. Units are in meters/second.

Example: 3e8

Programmatic Use
Block Parameter: PropagationSpeed
Type:double
Values:positive scalar
Default:physconst("LightSpeed")

Data Types: double

Doppler Offset (Hz) — Doppler axis offset
0.0 (default) | scalar

Doppler axis offset, specified as a scalar. This property applies a frequency offset to the Doppler axis. Units are in Hz.

Programmatic Use
Block Parameter: DopplerOffset
Type:double
Values:scalar
Default:0

Data Types: double

Time Span (s) — Time span of display
0.100 (default) | positive scalar

Time span of the intensity display, specified as a positive scalar. Units are in seconds.

Programmatic Use
Block Parameter: TimeSpan
Type:double
Values:scalar
Default:0.1

Data Types: `double`

Operating Frequency (Hz) — Operating frequency
`300e6` (default) | positive scalar

Operating frequency, specified as a positive scalar. Units are in Hz.

Programmatic Use
Block Parameter:`OperatingFrequency`
Type:double
Values:scalar
Default:`300000000`

Data Types: `double`

Doppler Resolution (Hz) — Doppler interval between samples
`1.0` (default) | positive scalar

Doppler interval between samples, specified as a positive scalar. This property defines the Doppler frequency difference between the scope columns. Units are in Hz.

Programmatic Use
Block Parameter:`DopplerResolution`
Type:double
Values:positive scalar
Default:`1`

Data Types: `double`

Time Resolution (s) — Time difference between rows
`0.001` (default) | positive scalar

Time interval between samples, specified as a positive scalar. This property defines the time duration between rows of scope. Units are in seconds.

Programmatic Use
Block Parameter:`TimeResolution`
Type:double
Values:positive scalar
Default:`0.001`

Data Types: `double`

Show Grid — Show Cartesian grid overlay
on (default) | off

Show Cartesian grid overlay. To show the grid, select the check box.

Programmatic Use
Block Parameter:`ShowGrid`
Type:logical
Values:1 | 0
Default:`1`

Show Ticks — Show tick value labels
on (default) | off

Show value labels on tick marks. The display the labels on, select the check box.

Programmatic Use

Block Parameter:`ShowTicks`
Type:logical
Values:1 | 0
Default:`1`

Labels

Time Label — Time axis label
`'Time (s)'` (default) | string

Time-axis label, specified as a string.

Tunable: Yes

Programmatic Use

Block Parameter:`TimeLabel`
Type:string
Values:string
Default:`'Time History (s)'`

Data Types: `string`

Colorbar Label — Color bar label
`'dB'` (default) | string

Color bar label, specified as a string.

Tunable: Yes

Programmatic Use

Block Parameter:`ColorBarLabel`
Type:string
Values:string
Default:`'dB'`

Data Types: `string`

Title — Window title
`'Doppler vs. Time'` (default) | string

Window title, specified as a string.

Tunable: Yes

Programmatic Use

Block Parameter:`Title`
Type:string
Values:string
Default:`'Doppler vs. Time'`

Data Types: `char`

Block Characteristics

Data Types	`double` | `single`

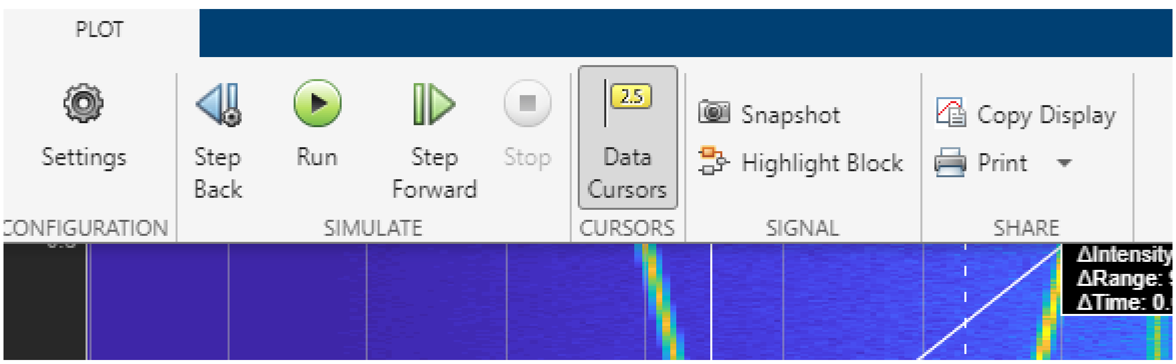
Direct Feedthrough	no
Multidimensional Signals	no
Variable-Size Signals	no
Zero-Crossing Detection	no

Algorithms

This section shows how to run the scope, configure the settings to show labels for the various features and use the cursor to examine data.

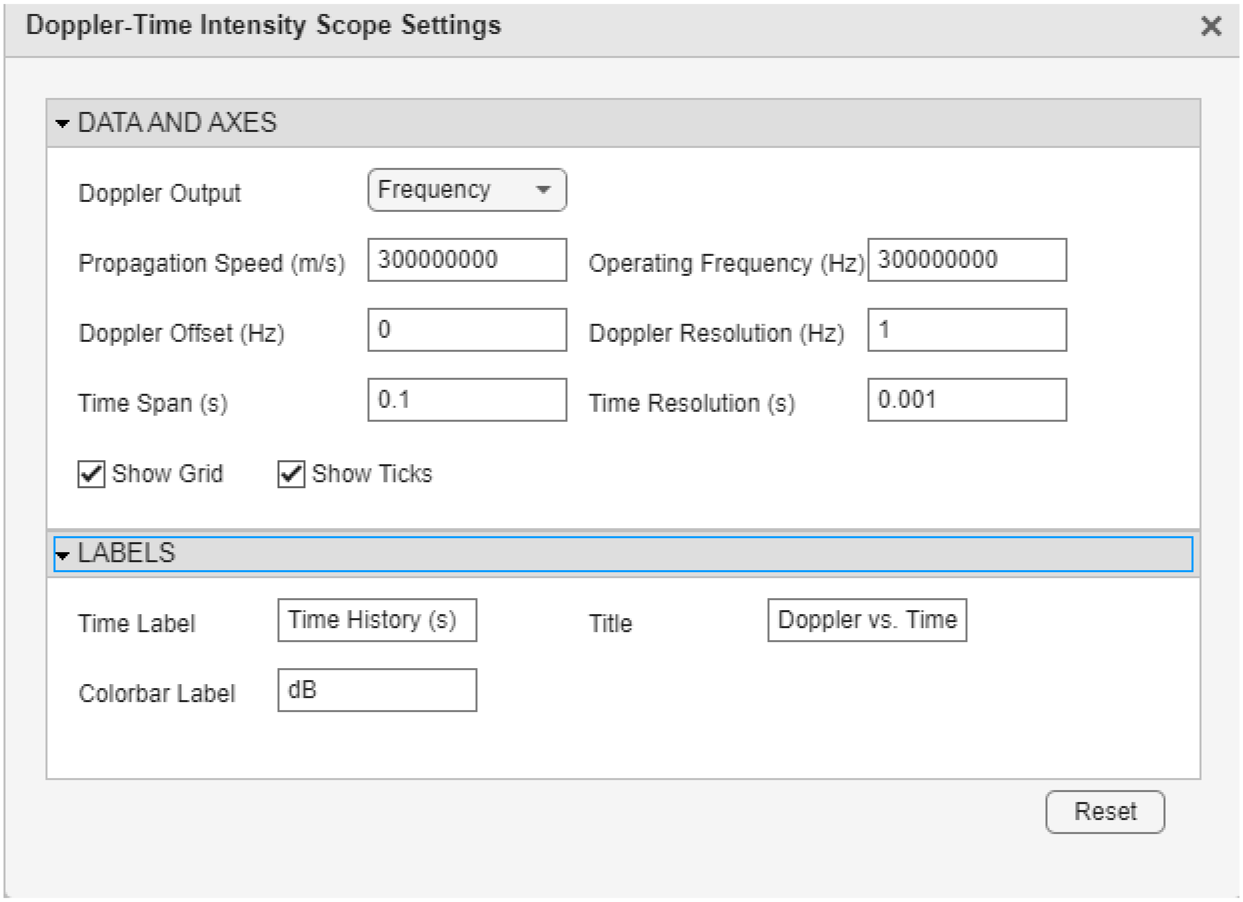
Run the Scope

After selecting and opening the block, the scope displays a single tab **PLOT**. Opening this tab shows all the controls available for the block. The **Settings** menu displays all the settable parameters for the scope. The **Step Back**, **Run**, **Step Forward**, and **Stop** buttons control the simulation. The **Data Cursors** button lets you examine data values on the display.



Scope Parameter Settings

Selecting the **Settings** menu from **PLOT** panel scope opens the parameter window with the **DATA AND AXES** and **LABELS** panels.

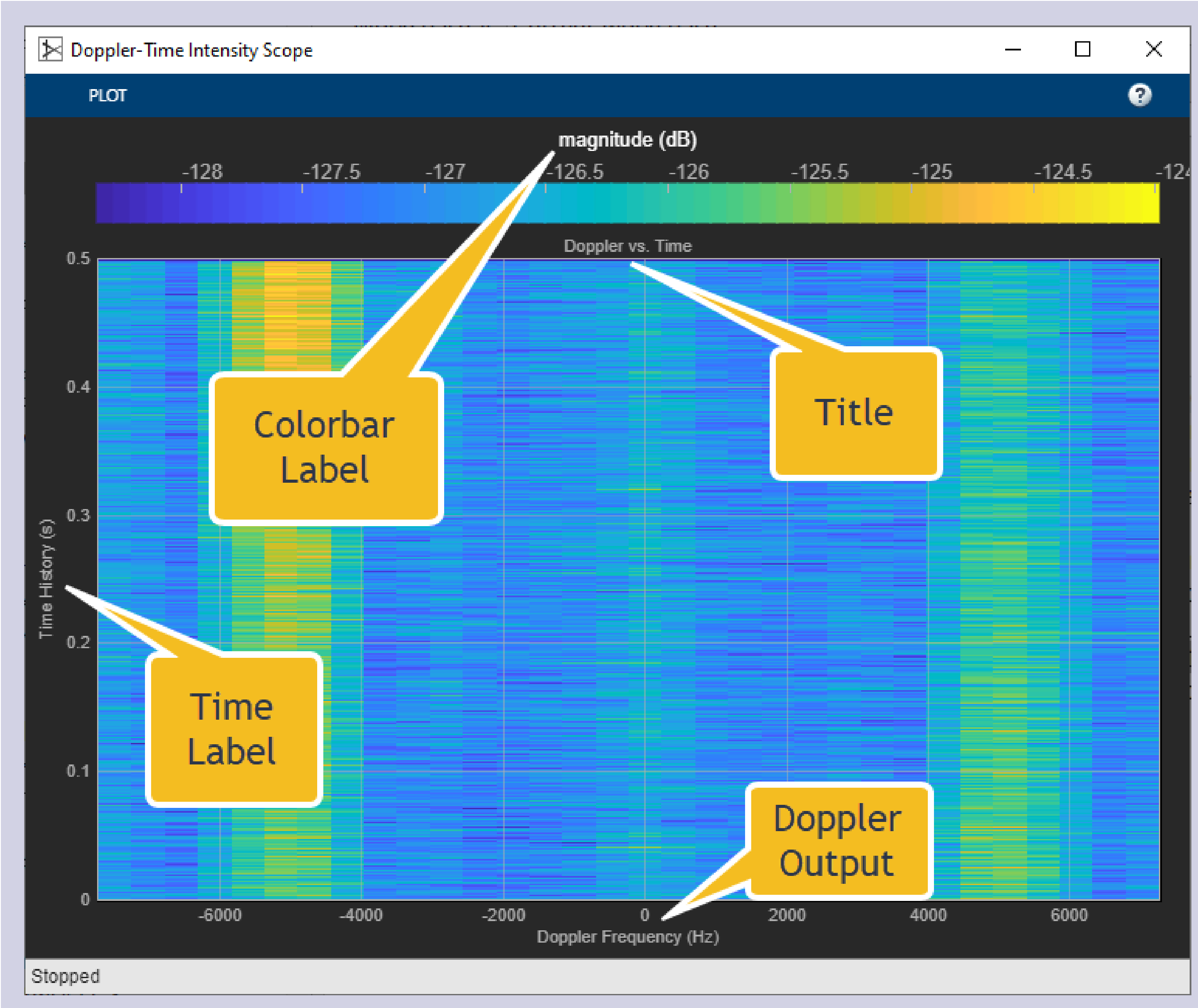


DATA AND AXES

- **Doppler Output** – Use this field to choose `Frequency` for the Doppler shift or `Speed` for the radial speed.
- **Operating Frequency (Hz)** - Use this field to set the scaling on the x-axis.
- **Doppler Offset (Hz)** - Use this field to set the frequency offset of the x-axis.
- **Doppler Resolution (Hz)** - Use this field to set the granularity of the x-axis.
- **Time Span (s)** - Use this field to set the duration of the y-axis.
- **Time Resolution (s)** - Use this field to set the granularity of the y-axis.

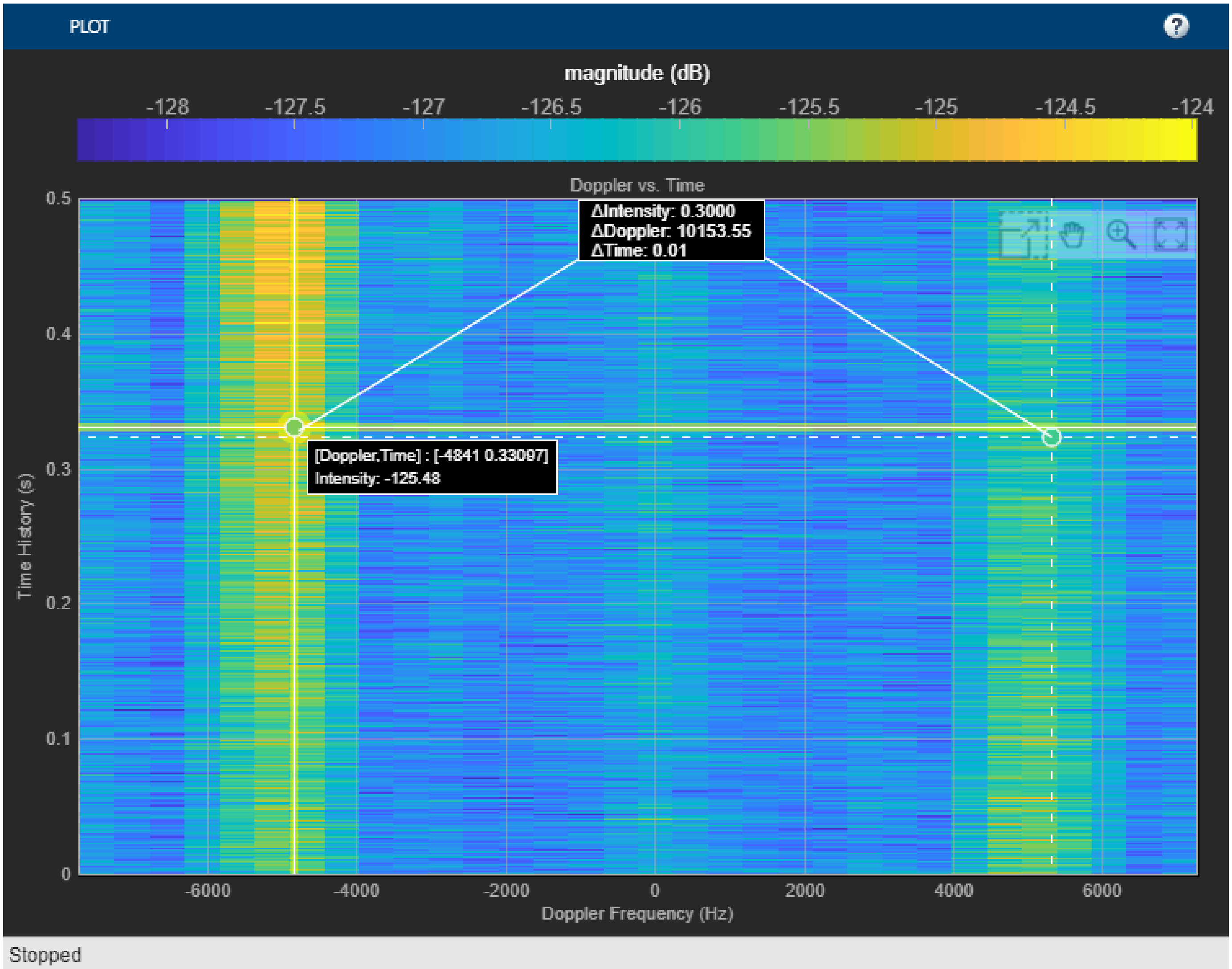
Labels

- **Time Label** – Use this field to set the y-axis label.
- **Title** – Use this field to set the display title.
- **Colorbar Label** – Set this field to set the color bar label.



Using Cursors

Select the **Cursors Button** to display the screen cursors. Each cursor consists of intersecting horizontal and vertical cursors that define a point on the display. Positioning two cursors on the display shows the difference between the intensities at the two cursor points.



Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

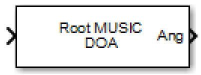
This block can be used for simulation visibility in systems that generate code but is not included in the generated code.

See Also

Range-Time Intensity Scope | Angle-Time Intensity Scope | `phased.DTIScope` | `phased.RTIScope`

Root MUSIC DOA

Root multiple signal classification (MUSIC) direction of arrival (DOA) estimator for ULA and UCA



Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The Root MUSIC DOA block estimates the direction of arrival of a specified number of narrowband signals incident on a uniform linear array or uniform circular array using the root multiple signal classification (Root MUSIC) algorithm.

Ports

Input

In — Input signal
complex-value matrix

Input signal, specified as a complex-value matrix. Matrix columns correspond to signal channels.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`

ElAng — Elevation angle
scalar

Elevation angle, specified as a scalar between -90° and 90°. The elevation angles for all signals must be the same as required by the phase mode excitation algorithm. Units are in degrees.

Dependencies

To enable this port, set the **Geometry** pull down menu in the `Sensor Array` panel to **UCA**.

Data Types: `single` | `double`

Output

Ang — Output
real-valued row vector

Estimated broadside direction-of-arrival angles, returned as real-valued row vector. Units are in degrees.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Number of signals — Number of signals

`1` (default) | positive integer scalar

Specify the number of signals as a positive integer scalar.

Forward-backward averaging — Enable forward-backward averaging
off (default) | on

Select this parameter to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold structure.

Spatial smoothing — Enable spatial smoothing
`0` (default) | non-negative integer

Specify the amount of averaging used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each increase in smoothing handles one extra coherent source, but reduces the effective number of elements by one. The maximum value of this parameter is $N - 2$, where N is the number of sensors in the ULA.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><th rowspan="2">Block Simulation</th><th colspan="3">Simulation Behavior</th></tr>
<tr><th><code>Normal</code></th><th><code>Accelerator</code></th><th><code>Rapid Accelerator</code></th></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use

Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cardioid Antenna` | `Cosine Antenna` | `Custom Antenna` | `Gaussian Antenna` | `Sinc Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cardioid Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Gaussian Antenna`

- `Sinc Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Null axis direction — Null axis direction
`-x` (default) | `+x` | `+y` | `-y` | `+z` | `-z`

Dependencies

To enable this parameter, set **Element type** to `Cardioid Antenna`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Align element normal with array normal — Align element normal with array normal
on (default) | off

Dependencies

This parameter is enabled when **Element type** is set to `Custom Antenna`.

Radiation pattern beamwidth (deg) — Radiation pattern beamwidth
`[10,10]` (default)

Dependencies

This parameter is enabled when **Element type** is set to `Gaussian Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Geometry

`ULA` (default) | `UCA`

Specify the array geometry as one of the following

- `ULA` — Uniform Line Array
- `UCA` — Uniform Circular Array

Number of elements — Number of array elements in U
`2` (default) | positive integer greater than or equal to two

The number of array elements for ULA arrays, specified as an integer greater than or equal to two.

Example: `11`

Data Types: `double`

Element spacing — Distance between ULA elements
`0.5` (default) | positive scalar

Distance between adjacent ULA elements, specified as a positive scalar. Units are in meters.

Example: `1.5`

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
`x`	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
`y`	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
`z`	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Taper — ULA array taper
`1` (default) | complex-valued vector

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Example: [0.5;1;0.5]

Data Types: `double`

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create a ULA array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.ULA('NumElements',13)`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Version History

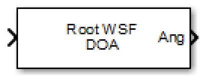
Introduced in R2014b

See Also

`phased.RootMUSICEstimator`

Root WSF DOA

Root weighted subspace fitting (WSF) direction of arrival (DOA) estimator for ULA



Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The Root WSF DOA block estimates the direction of arrival of a specified number of narrowband signals incident on a uniform linear array (ULA) using the root-weighted subspace fitting (RootWSF) algorithm.

Ports

Input

In — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* complex-valued matrix. *M* is the number of samples in the signals. The *N* columns of the matrix correspond to different signals.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `single` | `double`

Output

Ang — Estimated broadside DOA angles
P-length row vector

Estimated broadside direction-of-arrival (DOA) angles, returned as a *P*-length row vector. Units are in degrees.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Number of signals — Number of signals

`1` (default) | positive integer scalar

Specify the number of signals as a positive integer scalar.

Iterative method — Iterative method

`IMODE` (default) | `IQML`

Specify the iterative method as one of `IMODE` or `IQML`.

Maximum number of iterations — Maximum number of iterations

`Inf` (default)

Specify the maximum number of iterations as a positive integer or `Inf`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	Normal	Accelerator	Rapid Accelerator
Interpreted Execution	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
Code Generation	The block is compiled.	All blocks in the model are compiled.	

For more information, see "Choosing a Simulation Mode" (Simulink).

Programmatic Use

Block Parameter: SimulateUsing
Type:enum
Values: Interpreted Execution, Code Generation
Default: Interpreted Execution

Sensor Array Tab

Specify sensor array as — Method to specify array
Array (no subarrays) (default) | MATLAB expression

Method to specify array, specified as Array (no subarrays) or MATLAB expression.

- Array (no subarrays) — use the block parameters to specify the array.
- MATLAB expression — create the array using a MATLAB expression.

Element Parameters

Element type — Array element types
Isotropic Antenna (default) | Cardioid Antenna | Cosine Antenna | Custom Antenna | Gaussian Antenna | Sinc Antenna | Omni Microphone | Custom Microphone

Antenna or microphone type, specified as one of the following:

- Isotropic Antenna
- Cardioid Antenna
- Cosine Antenna
- Custom Antenna
- Gaussian Antenna
- Sinc Antenna
- Omni Microphone
- Custom Microphone

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
[0,1e20] (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Null axis direction — Null axis direction
`-x` (default) | `+x` | `+y` | `-y` | `+z` | `-z`

Dependencies

To enable this parameter, set **Element type** to `Cardioid Antenna`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-Q vector. Q must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Align element normal with array normal — Align element normal with array normal
on (default) | off

Dependencies

This parameter is enabled when **Element type** is set to `Custom Antenna`.

Radiation pattern beamwidth (deg) — Radiation pattern beamwidth
`[10,10]` (default)

Dependencies

This parameter is enabled when **Element type** is set to `Gaussian Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

ULA Parameters

Number of elements — Number of array elements in U
`2` (default) | positive integer greater than or equal to two

The number of array elements for ULA arrays, specified as an integer greater than or equal to two.

Example: `11`

Data Types: `double`

Element spacing — Distance between ULA elements
`0.5` (default) | positive scalar

Distance between adjacent ULA elements, specified as a positive scalar. Units are in meters.

Example: `1.5`

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Taper — ULA array taper
`1` (default) | complex-valued vector

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Example: [0.5;1;0.5]

Data Types: `double`

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create a ULA array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.ULA('NumElements',13)`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Version History

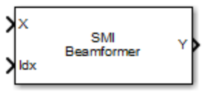
Introduced in R2014b

See Also

`phased.RootWSFEstimator`

SMI Beamformer

Sample matrix inversion (SMI) beamformer



Libraries:
Phased Array System Toolbox / Space-Time Adaptive Processing

Description

The SMI Beamformer block implements a sample matrix inversion (SMI) space-time adaptive beamformer employing the sample space-time covariance matrix.

Ports

Input

X — Input signal
M-by-*N*-by-*P* complex-valued matrix

Input signal, specified as an *M*-by-*N*-by-*P* complex-valued array. *M* is the number of range samples, *N* is the number of channels, and *P* is the number of pulses.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Idx — Index of range cells
positive integer

Index of range cells to compute processing weights.

Example: `1`

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, specified as a positive scalar.

Dependencies

To enable this port, set the **Specify PRF as** parameter to `Input port`.

Data Types: `double`

Ang — Targeting direction
2-by-1 real-valued vector

Targeting direction, specified as a 2-by-*1* real-valued vector. The vector takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie

between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Specify direction as** parameter to `Input port`.

Data Types: `double`

Dop — Targeting Doppler frequency
scalar

Targeting Doppler frequency of current pulse, specified as a scalar.

Dependencies

To enable this port, set the **Specify targeting Doppler as** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-1 complex-valued vector

Processing output, returned as an *M*-by-*1* complex-valued vector. The quantity *M* is the number of range samples in the input port X.

Data Types: `double`

W — Processing weights
length *N**P* complex-valued vector

Processing weights, returned as Length *N**P* complex-valued vector. The quantity *N* is the number of channels and *P* is the number of pulses. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays. *L* is the number of desired beamforming directions specified in the Ang input port or by the **Beamforming direction (deg)** parameter. There is one set of weights for each beamforming direction.

Dependencies

To enable this port, select the **Enable weights output** check box.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
3e8 (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Specify PRF as — Source of PRF value
Property (default) | Input port

Source of PRF value, specified as Property or Input port. When set to Property, the **Pulse repetition frequency (Hz)** parameter sets the PRF. When set to Input port, pass in the PRF using the PRF input port.

Pulse repetition frequency (Hz) — Pulse repetition frequency
1 (default) | positive scalar

Pulse repetition frequency, PRF, specified as a positive scalar. Units are in Hertz. Set this parameter to the same value set in any Waveform library block used in the simulation.

Dependencies

To enable this parameter, set the **Specify PRF as** parameter to Property.

Specify direction as — Specify source of targeting directions
Property (default) | Input port

Specify whether the targeting direction for the STAP processor block comes from a block parameter or from the ANG input port. Values of this parameter are

Property	• For the ADPCA Canceller and DPCA Canceller blocks, targeting direction is specified using **Receiving mainlobe direction (deg)**. • For the SMI Beamformer block, targeting direction is specified using **Targeting direction**. These parameters appear only when the **Specify direction as** parameter is set to Property.
Input port	Enter the targeting directions using the Ang input port. This port appears only when **Specify direction as** is set to Input port.

Targeting direction (deg) — Processor targeting direction
[0;0] (default) | real-valued length-2 column vector

Processor targeting direction, specified as a real-valued length-2 column vector of azimuth and elevation angles, [AzimuthAngle;ElevationAngle]. The azimuth angle is between –180° and 180° and the elevation angle is between –90° and 90°. Units are in degrees.

Dependencies

To enable this parameter, set **Specify direction as** to Property.

Number of bits in phase shifters — Number of phase shift quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Specify targeting Doppler as — Source of targeting Doppler
`Property` (default) | `Input port`

Specify whether targeting Doppler values for the STAP processor comes from the **Targeting Doppler (Hz)** parameter of this block or using the `DOP` input port. For the ADPCA Canceller and DPCA Canceller blocks, the **Specify targeting Doppler as** parameter appears only when the **Output pre-Doppler result** check box is cleared. Values of this parameter are

`Property`	Specify targeting Doppler values using the **Targeting Doppler** parameter of the block. The **Targeting Doppler** parameter appears only when **Specify targeting Doppler as** is set to `Property`.
`Input port`	Specify targeting Doppler values using the `Dop` input port. This port appears only when **Specify targeting Doppler as** is set to `Input port`.

Targeting Doppler (Hz) — Targeting Doppler of STAP processor
`0` (default) | scalar

Targeting Doppler of STAP processor, specified as a scalar.

Dependencies

- To enable this parameter for the SMI Beamformer block, set **Specify targeting Doppler as** to `Property`.
- To enable this parameter for the ADPCA Canceller and DPCA Canceller blocks, first clear the **Output pre-Doppler result** check box. Then set the **Specify targeting Doppler as** parameter to `Property`.

Number of guard cells — Number of guard cells using for training
2 (default) | positive even integer

Number of guard cells used for training, specified as a positive, even integer. Whenever possible, the set of guard cells is equally divided into regions before and after the test cell.

Number of training cells — Number of cells used for training
2 (default) | positive even integer

Number of cells used for training, specified as a positive even integer. Whenever possible, the set of training cells is equally divided into regions before and after the test cell.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see "Choosing a Simulation Mode" (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.

- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

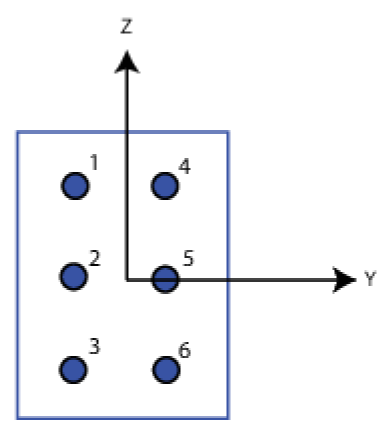
Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
`x`	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
`y`	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
`z`	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.

- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

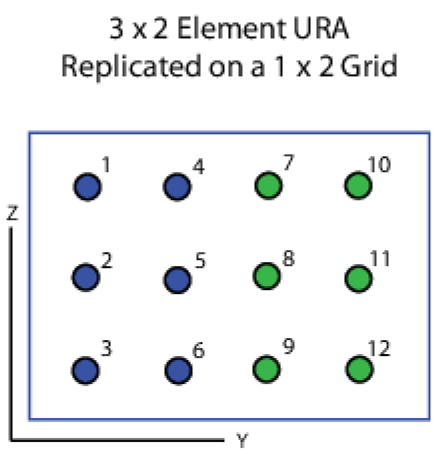
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.

- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

Introduced in R2014b

See Also

`phased.STAPSMIBeamformer`

Scattering MIMO Channel

Scattering MIMO propagation channel

Libraries:
Phased Array System Toolbox / Environment and Target

Description

The Scattering MIMO Channel models a 3-D multipath propagation channel in which radiated signals from a transmitting array are reflected from multiple scatters back towards a receiving array. In this channel, propagation paths are direct paths (line-of-sight) from point to point. The block models range-dependent time delay, gain, Doppler shift, phase change, and atmospheric loss due to gases, rain, fog, and clouds. You can optionally propagate a signal via a direct path from transmitter to receiver.

The attenuation models for atmospheric gases and rain are valid for electromagnetic signals in the frequency range 1–1000 GHz but the attenuation model for fog and clouds is valid for only 10–1000 GHz. Outside these frequency ranges, the object uses the nearest valid value.

Ports

Input

X — Transmitted narrowband signal
M-by-N_t complex-valued matrix

The transmitted narrowband signal, specified as an M-by-N_t complex-valued matrix. The quantity M is the number of samples in the signal, and N_t is the number of transmitting array elements. Each column represents the signal transmitted by the corresponding array element.

Example: `[1,1;j,1;0.5,0]`

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this port, set the **Polarization configuration** parameter to `None` or `Combined`.

Data Types: `double`
Complex Number Support: Yes

XH — Transmitted narrowband H-polarization signal
M-by-N_t complex-valued matrix

Transmitted narrowband H-polarization signal, specified as an M-by-N_t complex-valued matrix. The quantity M is the number of samples in the signal, and N_t is the number of transmitting array elements. Each column represents the signal transmitted by the corresponding array element.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Example: `[1,1;j,1;0.5,0]`

Dependencies

To enable this port, set the **Polarization configuration** parameter to `Dual`.

Data Types: `double`
Complex Number Support: Yes

XV — Transmitted narrowband V-polarization signal
M-by-N_t complex-valued matrix

Transmitted narrowband *V*-polarization signal, specified as an M-by-N_t complex-valued matrix. The quantity M is the number of samples in the signal, and N_t is the number of transmitting array elements. Each column represents the signal transmitted by the corresponding array element.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this port, set the **Polarization configuration** parameter to `Dual`.

Data Types: `double`
Complex Number Support: Yes

TxPos — Position of transmitting antenna array
3-by-1 real-valued column vector

Position of transmitting antenna array, specified as a 3-by-1 real-valued column vector taking the form `[x;y;z]`. The vector elements correspond to the *x*, *y*, and *z* positions of the array. Units are in meters.

Dependencies

To enable this port, set the **Source of transmit array motion** parameter to `Input port`.

Data Types: `double`

TxVel — Velocity of transmitting antenna array
3-by-1 real-valued column vector

Velocity of transmitting antenna array, specified as a 3-by-1 real-valued column vector taking the form `[vx;vy;vz]`. The vector elements correspond to the *x*, *y*, and *z* velocities of the array. Units are in meters per second.

Dependencies

To enable this port, set the **Source of transmit array motion** parameter to `Input port`.

Data Types: `double`

TxAxes — Axes orientation of transmitting antenna array
3-by-3 real orthonormal matrix

Axes orientation of transmitting antenna array, specified as a 3-by-3 real orthonormal matrix. The matrix defines the orientation of the array local coordinate system with respect to the global coordinates. Matrix columns correspond to the directions of the *x*, *y*, and *z* axes of the local coordinate system. Units are dimensionless.

Dependencies

To enable this port, set the **Source of transmit array motion** parameter to `Input port`.

Data Types: `double`

RxPos — Position of receiving antenna array
3-by-1 real-valued column vector

Position of receiving antenna array, specified as a 3-by-1 real-valued column vector taking the form `[x;y;z]`. The vector elements correspond to the *x*, *y*, and *z* positions of the array. Units are in meters.

Dependencies

To enable this port, set the **Source of receive array motion** parameter to `Input port`.

Data Types: `double`

RxVel — Velocity of receiving antenna array
3-by-1 real-valued column vector

Velocity of receiving antenna array, specified as a 3-by-1 real-valued column vector taking the form `[vx;vy;vz]`. The vector elements correspond to the *x*, *y*, and *z* velocities of the array. Units are in meters per second.

Dependencies

To enable this port, set the **Source of receive array motion** parameter to `Input port`.

Data Types: `double`

RxAxes — Axes orientation of receiving antenna array
3-by-3 real orthonormal matrix

Axes orientation of receiving antenna array, specified as a 3-by-3 real orthonormal matrix. The matrix defines the orientation of the array local coordinate system with respect to the global coordinates. Matrix columns correspond to the directions of the *x*, *y*, and *z* axes of the local coordinate system. Units are dimensionless.

Dependencies

To enable this port, set the **Source of receive array motion** parameter to `Input port`.

Data Types: `double`

ScatPos — Positions of scatterers
3-by-N_s real-valued matrix

Position of scatterers, specified as a 3-by-N_s real-valued matrix. Each column of the matrix takes the form `[x;y;z]`, containing the *x*, *y*, and *z* positions of a scatterer. Units are in meters.

Dependencies

To enable this port, set the **Scatterer specification** parameter to `Input port`.

Data Types: `double`

ScatVel — Velocities of scatterers
3-by-N_s real-valued matrix

Velocities of scatterers, specified as a 3-by-N_s real-valued matrix. Each matrix column has the form `[vx;vy;vz]`, containing the *x*, *y*, and *z* velocities of a scatterer. Units are in meters per second.

Dependencies

To enable this port, set the **Scatterer specification** parameter to `Input port`.

Data Types: `double`

ScatCoef — Scattering coefficients
1-by-N_s complex-valued row vector

Scattering coefficients, specified as a 1-by-N_s complex-valued row vector. Each vector element specifies the scattering coefficient of the corresponding scatterer. Units are dimensionless.

Dependencies

To enable this port, set the **Scatterer specification** parameter to `Input port`.

Data Types: `double`

Output

Y — Received narrowband signal
M-by-N_r complex-valued matrix

Received narrowband signal, returned as an M-by-N_r complex-valued matrix. The quantity M is the number of samples in the signal, and N_r is the number of receiving array elements. Each column represents the signal received by the corresponding array element.

Dependencies

To enable this port, set the **Polarization configuration** parameter to `None` or `Combined`.

Data Types: `double`
Complex Number Support: Yes

YH — Received narrowband H-polarization signal
complex-valued M-by-N_r matrix

Received narrowband *H*-polarization signal, returned as a complex-valued M-by-N_r matrix. M is the number of samples in the signal, and N_r is the number of receiving array elements. Each column represents the signal received by the corresponding array element.

Dependencies

To enable this port, set the **Polarization configuration** parameter to `Dual`.

Data Types: `double`
Complex Number Support: Yes

YV — Received narrowband V-polarization signal
complex-valued M-by-N_r matrix

Received narrowband V-polarization signal, returned as a complex-valued M-by-N_r matrix. M is the number of samples in the signal, and N_r is the number of receiving array elements. Each column represents the signal received by the corresponding array element.

Dependencies

To enable this port, set the **Polarization configuration** parameter to `Dual`.

Data Types: `double`
Complex Number Support: Yes

CS — Channel response
N_t-by-N_r-by-N_s complex-valued MATLAB array

Channel response, returned as an N_t-by-N_r-by-N_s complex-valued MATLAB array. N_t is the number of transmitting array elements. N_r is the number of receiving array elements. N_s is the number of scatterers. Each page of the array corresponds to the channel response matrix for a specific scatterer.

Dependencies

To enable this port, select the **Output channel response** check box.

Data Types: `double`
Complex Number Support: Yes

Tau — Path delays
1-by-N_s real-valued vector

Path delays, returned as a 1-by-N_s real-valued vector. N_s is the number of scatterers. Each element corresponds to the path time delay from the transmitting array phase center to the scatterer and then to the receiving array phase center.

Dependencies

To enable this port, select the **Output channel response** checkbox.

Data Types: `double`

Parameters

Main Tab

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`.

Data Types: `double`

Signal carrier frequency (Hz) — Signal carrier frequency
`3e8` (default) | positive real-valued scalar

Signal carrier frequency, specified as a positive real-valued scalar. Units are in hertz.

Data Types: `double`

Polarization configuration — Polarization configuration
`None` (default) | `Combined` | `Dual`

Polarization configuration, specified as `None`, `Combined`, or `Dual`. When you set this parameter to `None`, the output field is considered a scalar field. When you set this parameter to `None`, the radiated fields are polarized and are interpreted as a single signal in the sensor's inherent polarization. When you set this parameter to `Dual`, the *H* and *V* polarization components of the radiated field are independent signals.

Data Types: `char`

Specify atmospheric parameters — Enable atmospheric attenuation model
off (default) | on

Select this parameter to enable to add signal attenuation caused by atmospheric gases, rain, fog, or clouds. When you select this parameter, the **Temperature (degrees Celsius)**, **Dry air pressure (Pa)**, **Water vapour density (g/m^3)**, **Liquid water density (g/m^3)**, and **Rain rate (mm/hr)** parameters appear in the dialog box.

Data Types: `Boolean`

Temperature (degrees Celsius) — Ambient temperature
`15` (default) | real-valued scalar

Ambient temperature, specified as a real-valued scalar. Units are in degrees Celsius.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `double`

Dry air pressure (Pa) — Atmospheric dry air pressure
`101.325e3` (default) | positive real-valued scalar

Atmospheric dry air pressure, specified as a positive real-valued scalar. Units are in pascals (Pa). The default value of this parameter corresponds to one standard atmosphere.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `double`

Water vapour density (g/m^3) — Atmospheric water vapor density
`7.5` (default) | positive real-valued scalar

Atmospheric water vapor density, specified as a positive real-valued scalar. Units are in g/m^3.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `datetime`

Liquid water density (g/m^3) — Liquid water density
`0.0` (default) | nonnegative real-valued scalar

Liquid water density of fog or clouds, specified as a nonnegative real-valued scalar. Units are in g/m^3. Typical values for liquid water density are 0.05 for medium fog and 0.5 for thick fog.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `double`

Rain rate (mm/hr) — Rainfall rate
`0.0` (default) | non-negative real-valued scalar

Rainfall rate, specified as a nonnegative real-valued scalar. Units are in mm/hr.

Dependencies

To enable this parameter, select the **Specify atmospheric parameters** checkbox.

Data Types: `double`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Simulate direct path propagation — Enable propagation along direct path
off (default) | on

Select this check box to enable signal propagation along the line-of-sight direct path from the transmitting array to the receiving array with no scattering.

Data Types: Boolean

Maximum delay (s) — Maximum signal delay
`10e-6` (default) | positive scalar

The maximum signal delay, specified as a positive scalar. Delays greater than this value are ignored.

Data Types: double

Output channel response — Enable output of channel response
off (default) | on

Select this checkbox to output the channel response and time delay via the output ports `CS` and `Tau`.

Data Types: Boolean

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see "Choosing a Simulation Mode" (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum

Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Transmit and Receive Array Tabs

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
on (default) | off

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees

azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

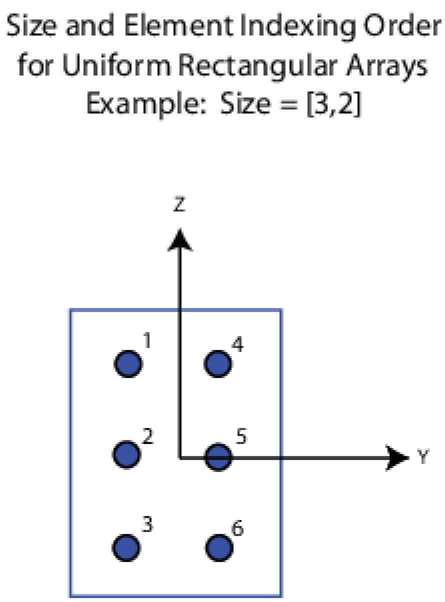
- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of elements in each row and column.

For a URA, array elements are indexed from top to bottom along the leftmost array column, and continued to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to URA.

Array normal — Array normal direction
x for URA arrays or z for UCA arrays (default) | y

Array normal direction, specified as x, y, or z.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Data Types: `double`

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. If the parameter value is a matrix, each

column specifies the normal direction of the corresponding element in the form [azimuth;elevation] with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

To enable this parameter, set **Geometry** to `Conformal Array`.

Data Types: `double`

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Motion Tab

Source of transmit array motion — Specify the source of the transmitting array motion parameters
`Property` (default) | `Input port`

Source of transmitting array motion parameters, specified as `Property` or `Input port`.

- When you select `Property`, specify the array location and orientation using the **Position of the transmit array (m)** and **Orientation of the transmit array** parameters. The array is stationary.
- When you select `Input port`, specify the array location, velocity, and orientation using the `TxPos`, `TxVel`, and `TxAxes` input ports of the block.

Data Types: `char`

Position of the transmit array (m) — Position of transmitting array
`[0;0;0]` (default) | real-valued 3-by-1 vector

The position of the transmitting array phase center, specified as a real-valued, 3-by-1 vector in Cartesian form `[x;y;z]` with respect to the global coordinate system. Units are in meters.

Dependencies

To enable this parameter, set the **Source of transmit array motion** parameter to `Property`.

Data Types: `double`

Orientation of the transmit array — Set the orientation of transmitting array axes
`eye(3,3)` (default) | real-valued 3-by-3 orthonormal matrix

The orientation of transmitting array, specified as a real-valued, 3-by-3 orthonormal matrix. The matrix specifies the directions of the three axes that define the local coordinate system of the array with respect to the global coordinate system. The columns of the array correspond to the *x*, *y*, and *z* axes, respectively.

Dependencies

To enable this parameter, set the **Source of transmit array motion** parameter to `Property`.

Data Types: `double`

Source of receive array motion — Specify the source of the receiving array motion parameters
`Property` (default) | `Input port`

Source of receiving array motion parameters, specified as `Property` or `Input port`.

- When you select `Property`, specify the array location and orientation using the **Position of the receive array (m)** and **Orientation of the receive array** parameters. The array is stationary.
- When you select `Input port`, specify the array location, velocity, and orientation using the `RxPos`, `RxVel`, and `RxAxes` input ports of the block.

Data Types: `char`

Position of the receive array (m) — Position of receiving array
`[physconst('LightSpeed' )/1e5; 0;0]` (default) | real-valued 3-by-1 vector

The position of the receiving array phase center, specified as a real-valued, 3-by-1 vector in Cartesian form `[x;y;z]` with respect to the global coordinate system. Units are in meters.

Dependencies

To enable this parameter, set the **Source of receive array motion** parameter to `Property`.

Data Types: `double`

Orientation of the receive array — Set the orientation of receiving array axes
`eye(3,3)` (default) | real-valued 3-by-3 orthonormal matrix

The orientation of receiving array, specified as a real-valued, 3-by-3 orthonormal matrix. The matrix specifies the directions of the three axes that define the local coordinate system of the array with respect to the global coordinate system. The columns of the array correspond to the *x*, *y*, and *z* axes, respectively.

Dependencies

To enable this parameter, set the **Source of receive array motion** parameter to `Property`.

Data Types: `double`

Scatterer specification — Specify source of scatterer parameters
`Auto` (default) | `Property` | `Input port`

The source of scatterer parameters, specified as `Auto`, `Property`, or `Input port`.

- When you set this parameter to `Auto`, all scatterer positions and coefficients are randomly generated. Scatterer velocities are zero. The generated positions are contained within the region set by the **Boundary of scatterer positions** parameter. Set the number of scatterers using the **Number of scatterers** parameter.
- When you set this property to `Property`, set the scatterer positions using the **Positions of scatterers (m)** parameter. Set the scattering coefficients using the **Scattering coefficients** parameter. Scatterer velocities are zero.
- When you set this parameter to `Input port`, you specify the scatterer positions, velocities, and scattering coefficients using the `ScatPos`, `ScatVel`, and `ScatCoef` block input ports.

Data Types: `char`

Number of scatterers — Number of scatterers
`1` (default) | nonnegative integer

The number of scatterers, specified as a nonnegative integer.

Dependencies

To enable this property, set the **Scatterer specification** parameter to `Auto`.

Data Types: `double`

Boundary of scatterer positions — Constrain scatterer positions within a boundary
`[0,1000]` (default) | 1-by-2 real-valued vector | 3-by-2 real-valued matrix

The boundary scatterer positions, specified as a 1-by-2 real-valued row vector or a 3-by-2 real-valued matrix. If the boundary is a 1-by-2 row vector, the vector contains the minimum and maximum, `[minbdry maxbdry]`, for all three dimensions. If the boundary is a 3-by-2 matrix, the matrix specifies boundaries in all three dimensions in the form `[x_minbdry x_maxbdry;y_minbdry y_maxbdry; z_minbdry z_maxbdry]`.

Dependencies

To enable this property, set the **Scatterer specification** parameter to `Auto`.

Data Types: `double`

Positions of scatterers (m) — Positions of scatterers
`[physconst('LightSpeed' )*5e-6;0;0]` (default) | real-valued 3-by-N_s matrix

The positions of the scatterers, specified as real-valued 3-by-N_s matrix. N_s is the number of scatterers. Each column represents a different scatterer and has the Cartesian form `[x;y;z]` with respect to the global coordinate system. Units are in meters.

Dependencies

To enable this property, set the **Scatterer specification** parameter to `Property`.

Data Types: `double`

ScattererCoefficient — Scattering coefficients
1 (default) | complex-valued 1-by-N_s matrix

Scattering coefficients, specified as a complex-valued 1-by-N_s vector. N_s is the number of scatterers. Units are dimensionless.

Dependencies

To enable this property, set the **Scatterer specification** parameter to `Property`.

Data Types: `double`

Version History

Introduced in R2017a

See Also

Objects
`phased.ScatteringMIMOChannel`

Stepped FM Waveform

Stepped FM pulse waveform

Libraries:
Phased Array System Toolbox / Waveforms

Description

The Stepped FM Waveform block generates a stepped FM pulse waveform with a specified pulse width, pulse repetition frequency (PRF), and number of frequency steps. The transmitted frequency is incremented in constant steps over the duration of the pulse. The block outputs an integral number of pulses or samples.

Ports

Input

PRFIdx — PRF Index
positive integer

Index to select the pulse repetition frequency (PRF), specified as a positive integer. The index selects the PRF from the predefined vector of values specified by the **Pulse repetition frequency (Hz)** parameter.

Example: 4

Dependencies

To enable this port, select **Enable PRF selection input**.

Data Types: `double`

FreqOffset — Frequency offset
scalar

Frequency offset in Hz, specified as a scalar.

Example: `2e3`

Dependencies

To enable this port, set **Source of Frequency Offset** to `Input port`.

Data Types: `double`

Output

Y — Pulse waveform
complex-valued vector

Pulse waveform samples, returned as a complex-valued vector.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, returned as a positive scalar.

Dependencies

To enable this port, set the **Output signal format** parameter to `Pulses` and then select the **Enable PRF output** parameter.

Data Types: `double`

Coeff — Matched filter coefficients
vector | matrix

Matched filter coefficients, returned as a vector or matrix.

Dependencies

To enable this port, select **Enable Matched Filter Coefficients Output**.

Data Types: `double`

Parameters

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.

Programmatic Use
Block Parameter:`SampleRate`
Type:double
Values:positive scalar
Default:`1e6`

Method to specify pulse duration — Pulse duration as time or duty cycle
`Pulse width` (default) | `Duty cycle`

Method to set the pulse duration, specified as `Pulse width` or `Duty cycle`. When you set this parameter to `Pulse width`, the pulse duration is set using the **Pulse width (s)** parameter. When you set this parameter to `Duty cycle`, the pulse duration is computed from the values of the **Pulse repetition frequency (Hz)** and **Duty Cycle** parameters.

Programmatic Use
Block Parameter:`DurationSpecification`
Type:string
Values:string
Default:`'Pulse width'`

Pulse width (s) — Time duration of pulse
`50e-6` (default) | positive scalar

The duration of each pulse, specified as a positive scalar. Set the product of **Pulse width (s)** and **Pulse repetition frequency** to be less than or equal to one. This restriction ensures that the pulse width is smaller than the pulse repetition interval. Units are in seconds.

Example: `300e-6`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Pulse width`.

Programmatic Use
Block Parameter:`PulseWidth`
Type:double
Values:string
Default:`50e-6`

Duty cycle — Waveform duty cycle
`0.5` (default) | scalar in the range *[0,1]*

Waveform duty cycle, specified as a scalar in the range *[0,1]*.

Example: `0.7`

Dependencies

To enable this parameter, set the **Method to specify pulse duration** parameter to `Duty cycle`.

Programmatic Use
Block Parameter:`DutyCycle`
Type:double
Values:positive scalar
Default:`1e6`

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1e4` (default) | positive scalar

Pulse repetition frequency, *PRF*, specified as a scalar or a row vector. Units are in Hz. The pulse repetition interval, *PRI*, is the inverse of the pulse repetition frequency, *PRF*. The value of **Pulse repetition frequency (Hz)** must satisfy these constraints:

- The product of **Pulse width** and **Pulse repetition frequency (Hz)** must be less than or equal to one. This condition expresses the requirement that the pulse width is less than one pulse repetition interval. For the phase-coded waveform, the pulse width is the product of the chip width and number of chips.
- The ratio of sample rate to any element of **Pulse repetition frequency** must be an integer. This condition expresses the requirement that the number of samples in one pulse repetition interval is an integer.

You can select the value of *PRF* by using block parameter settings alone or in conjunction with the input port, `PRFIdx`.

- When the **Enable PRF selection input** parameter is not selected, set the *PRF* using block parameters.
  - To implement a constant *PRF*, specify **Pulse repetition frequency (Hz)** as a positive scalar.
  - To implement a staggered *PRF*, specify **Pulse repetition frequency (Hz)** as a row vector with positive values. After the waveform reaches the last element of the vector, the process continues cyclically with the first element of the vector. When *PRF* is staggered, the time between successive output pulses cycles through the successive values of the *PRF* vector.
- When the **Enable PRF selection input** parameter is selected, you can implement a selectable *PRF* by specifying **Pulse repetition frequency (Hz)** as a row vector with positive real-valued entries. But this time, when you execute the block, select a *PRF* by passing an index into the *PRF* vector into the `PRFIdx` port.

In all cases, the number of output samples is fixed when you set the **Output signal format** to `Samples`. When you use a varying *PRF* and set **Output signal format** to `Pulses`, the number of output samples can vary.

Programmatic Use
Block Parameter:PRF
Type:double
Values:positive scalar
Default:`1e6`

Enable PRF selection input — Select predefined PRF
off (default) | on

Select this parameter to enable the `PRFIdx` port.

- When enabled, pass in an index into a vector of predefined PRFs. Set predefined PRFs using the **Pulse repetition frequency (Hz)** parameter.
- When not enabled, the block cycles through the vector of PRFs specified by the **Pulse repetition frequency (Hz)** parameter. If **Pulse repetition frequency (Hz)** is a scalar, the PRF is constant.

Programmatic Use
Block Parameter:`PRFSelectionInputPort`
Type:logical
Values:positive scalar
Default:`off`

Frequency step (Hz) — Linear frequency step size
`2e4` (default) | positive scalar

Specify the linear frequency step size as a positive scalar. Units are in Hertz.

Example: `1e3`

Number of frequency steps — Number of frequency steps in pulse
5 (default) | positive integer

Specify the number of frequency steps as a positive integer. When the **Number of frequency steps** is 1, the stepped FM waveform reduces to a rectangular waveform.

Example: 8

Source of simulation sample time — Source of simulation sample time
`Derive from waveform parameters` (default) | `Inherit from Simulink engine`

Source of simulation sample time, specified as `Derive from waveform parameters` or `Inherit from Simulink engine`. When set to `Derive from waveform parameters`, the block runs at a variable rate determined by the PRF of the selected waveform. The elapsed time is variable. When set to `Inherit from Simulink engine`, the block runs at a fixed rate so the elapsed time is a constant.

Dependencies

To enable this parameter, select the **Enable PRF selection input** parameter.

Programmatic Use
Block Parameter:`SimulationTimeSource`
Type:enum
Values:`Derive from waveform parameters`, `Inherit from Simulink engine`
Default:`Derive from waveform parameters`

Source of Frequency Offset — Source of frequency offset
`Property` (default) | `Input port`

Source of frequency offset, specified as `Property` or `Input port`.

- When set to `Property`, the offset is determined by the value of the **Frequency Offset** parameter.
- When set to `Input port`, the offset is determined by the value of the **FreqOffset** port.

Programmatic Use
Block Parameter:`FrequencyOffsetSource`
Type:enum
Values:`Property`, `Input Port`
Default:`Property`

Frequency Offset (Hz) — Frequency offset
`0` (default) | scalar

Frequency offset, specified as a scalar. Units are in Hz.

Example: `2e3`

Dependencies

To enable this parameter set the **Source of Frequency Offset** parameter to `Input port`.

Programmatic Use
Block Parameter:`FrequencyOffset`
Type:double
Values:`scalar`
Default:`0`

Output signal format — Format of the output signal
`Pulses` (default) | `Samples`

The format of the output signal, specified as `Pulses` or `Samples`.

If you set this parameter to `Samples`, the output of the block consists of multiple samples. The number of samples is the value of the **Number of samples in output** parameter.

If you set this parameter to `Pulses`, the output of the block consists of multiple pulses. The number of pulses is the value of the **Number of pulses in output** parameter.

Programmatic Use
Block Parameter:`OutputFormat`
Type:enum
Values:`Pulses` `Samples`
Default:`Pulses`

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use
Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Number of pulses in output — Number of pulses in output
`1` (default) | positive integer

Number of pulses in the block output, specified as a positive integer.

Example: `2`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Pulses`.

Programmatic Use
Block Parameter:`NumPulses`
Type:double
Values:`positive scalar`
Default:`1`

Data Types: `double`

Enable PRF Output — Enable output of PRF
`off` (default) | `on`

Select this parameter to enable the PRF output port.

Dependencies

To enable this parameter, set **Output signal format** to `Pulses`.

Programmatic Use

Block Parameter:`PRFOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Enable Matched Filter Coeficients Output — Enable output of matched filter coefficients
`off` (default) | `on`

Select this parameter to enable the `Coeff` output port.

Programmatic Use

Block Parameter:`CoefficientOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see "Choosing a Simulation Mode" (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

Introduced in R2014b

See Also

`phased.SteppedFMWaveform`

Stretch Processor

Stretch processor for linear FM waveforms

Libraries:
Phased Array System Toolbox / Detection

Description

The Stretch Processor block applies stretch processing on a linear FM waveform. Also known as dechirping, stretch processing is an alternative to matched filtering for linear FM waveforms.

Ports

Input

X — Input signal
M-by-*P* complex-valued matrix

Input signal, specified as an *M*-by-*P* complex-valued array. *M* is the number of samples and *P* is the number of pulses.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

PRF — Pulse repetition frequency
positive scalar

Pulse repetition frequency of current pulse, specified as a positive scalar.

Dependencies

To enable this port, set the **Specify PRF as** parameter to `Input port`.

Data Types: `double`

Output

Y — Stretch processed signal
M-by-*P* complex-valued matrix

Stretch processed signal, returned as an *M*-by-*P* complex-valued array. *M* is the number of samples and *P* is the number of pulses.

Data Types: `double`

Parameters

Sample rate (Hz) — Sample rate of the output waveform
`1e6` (default) | positive scalar

Sample rate of the output waveform, specified as a positive scalar. The ratio of **Sample rate (Hz)** to each element in the **Pulse repetition frequency (Hz)** vector must be an integer. This restriction is equivalent to requiring that the pulse repetition interval is an integral multiple of the sample interval.

Programmatic Use
Block Parameter:`SampleRate`
Type:double
Values:positive scalar
Default:`1e6`

Pulse width (s) — Time duration of pulse
`50e-6` (default) | positive scalar

The duration of each pulse, specified as a positive scalar. Set the product of **Pulse width (s)** and **Pulse repetition frequency** to be less than or equal to one. This restriction ensures that the pulse width is smaller than the pulse repetition interval. Units are in seconds.

Example: `300e-6`

Specify PRF as — Source of PRF value
`Property` (default) | `Auto` | `Input port`

Source of PRF value for the stretch processor, specified as `Property`, `Auto`, or `Input port`. When set to `Property`, the **Pulse repetition frequency (Hz)** parameter sets the PRF. When set to `Input port`, pass in the PRF using the `PRF` input port. When set to `Auto`, PRF is computed from the number of rows in the input signal.

.

Pulse repetition frequency (Hz) — Pulse repetition frequency
`1e4` (default) | positive scalar

Pulse repetition frequency, PRF, specified as a positive scalar. Units are in Hertz. Set this parameter to the same value set in any `Waveform` library block used in the simulation.

Dependencies

To enable this parameter, set the **Specify PRF as** parameter to `Property`.

FM sweep slope (Hz/s) — Slope of linear FM sweep
`2e9` (default) | scalar

Slope of the linear FM sweeping as a scalar, specified as a scalar. Units are in Hertz per second.

Example: `1e3`

FM sweep interval — Direction of FM sweep
`Positive` (default) | `Symmetric`

FM sweep interval, specified as `Positive` or `Symmetric`. If you set this parameter value to `Positive`, the waveform sweeps the frequency bandwidth between *0* and *B*, where *B* is the frequency bandwidth. If you set this parameter value to `Symmetric`, the waveform sweeps in the interval between *–B/2* and *B/2*.

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Reference range (m) — Center of ranges of interest
`5000` (default) | positive scalar

Center of ranges of interest, specified as a positive scalar. The reference range must be within the unambiguous range of one pulse. Units are in meters.

Example: `10e3`

Reference span (m) — Span of ranges of interest
`500` (default) | positive scalar

Span of ranges of interest, specified as a positive scalar. The span of ranges is centered on the range specified by the **Reference range (m)** parameter. Units are in meters.

Example: `1e3`

Source of simulation sample time — Source of simulation sample time
`Derive from waveform parameters` (default) | `Inherit from Simulink engine`

Source of simulation sample time, specified as `Derive from waveform parameters` or `Inherit from Simulink engine`. When set to `Derive from waveform parameters`, the block runs at a variable rate determined by the PRF of the selected waveform. The elapsed time is variable. When set to `Inherit from Simulink engine`, the block runs at a fixed rate so the elapsed time is a constant.

Dependencies

To enable this parameter, select the **Enable PRF selection input** parameter.

Programmatic Use
Block Parameter:`SimulationTimeSource`
Type:enum
Values:`Derive from waveform parameters`, `Inherit from Simulink engine`

Default:`Derive from waveform parameters`

Output signal format — Format of the output signal
`Pulses` (default) | `Samples`

The format of the output signal, specified as `Pulses` or `Samples`.

If you set this parameter to `Samples`, the output of the block consists of multiple samples. The number of samples is the value of the **Number of samples in output** parameter.

If you set this parameter to `Pulses`, the output of the block consists of multiple pulses. The number of pulses is the value of the **Number of pulses in output** parameter.

Programmatic Use
Block Parameter:`OutputFormat`
Type:enum
Values:`Pulses` `Samples`
Default:`Pulses`

Number of samples in output — Number of samples in output
`100` (default) | positive integer

Number of samples in the block output, specified as a positive integer.

Example: `1000`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Samples`.

Programmatic Use
Block Parameter:`NumSamples`
Type:double
Values:`positive scalar`
Default:`100`

Data Types: `double`

Number of pulses in output — Number of pulses in output
`1` (default) | positive integer

Number of pulses in the block output, specified as a positive integer.

Example: `2`

Dependencies

To enable this parameter, set the **Output signal format** parameter to `Pulses`.

Programmatic Use
Block Parameter:`NumPulses`
Type:double
Values:`positive scalar`
Default:`1`

Data Types: `double`

Enable PRF Output — Enable output of PRF
`off` (default) | `on`

Select this parameter to enable the PRF output port.

Dependencies

To enable this parameter, set **Output signal format** to `Pulses`.

Programmatic Use
Block Parameter:`PRFOutputPort`
Type:enum
Values:`off` `on`
Default:`off`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

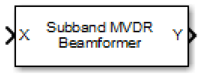
Introduced in R2014b

See Also

`phased.StretchProcessor`

Subband MVDR Beamformer

Subband MVDR (Capon) beamformer



Libraries:
Phased Array System Toolbox / Beamforming

Description

The Subband MVDR Beamformer block performs minimum variance distortionless response (MVDR) beamforming on wideband signals. Signals are decomposed into frequency subbands and narrowband MVDR beamforming is performed in each band. The resulting subband signals are summed to form the output signal. MVDR beamforming preserves signal power in a given direction while suppressing interference and noise from other directions. The MVDR beamformer is also called the Capon beamformer.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

XT — Training signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this port, select the **Enable training data input** check box.

Data Types: `double`

Ang — Beamforming direction
2-by-1 real-valued vector | 2-by-*L* real-valued matrix

Beamforming direction, specified as a 2-by-*L* real-valued matrix, where *L* is the number of beamforming directions. Each column takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Source of beamforming direction** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-*L* complex-valued matrix

Beamformed output, returned as an *M*-by-*L* complex-valued matrix. The quantity *M* is the number of signal samples and *L* is the number of desired beamforming directions specified by the `Beamforming direction` parameter or from the `Ang` port.

Data Types: `double`

Freq — Subband center frequencies
K-by-1 real-valued column vector

Subband center frequencies, returned as *K*-by-1 real-valued column vector. The quantity *K* is the number of subbands specified by the `Number of subbands` property.

Dependencies

To enable this port, select the **Enable subband center frequencies output** checkbox.

Data Types: `double`

W — Beamforming weights
N-by-*L* complex-valued matrix

Beamformed weights, returned as an *N*-by-*L* complex-valued matrix. The quantity *N* is the number of array elements. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays. *L* is the number of desired beamforming directions specified in the `Ang` port or by the `Beamforming direction (deg)` property. There is one set of weights for each beamforming direction.

Dependencies

To enable this port, select the **Enable weights output** checkbox.

Data Types: `double`

Parameters

Main tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Number of subbands — Number of processing subbands
`64` (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: `128`

Diagonal loading factor — Diagonal loading factor for stability
nonnegative scalar

Specify the diagonal loading factor as a nonnegative scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample support is small.

Enable training data input — Enable the use of training data
off (default) | on

Select this check box to specify additional training data via the input port XT. To use the input signal as the training data, clear the check box which removes the port.

Source of beamforming direction — Source of beamforming direction
`Property` (default) | `Input port`

Source of beamforming direction, specified as `Property` or `Input port`. When you set **Source of beamforming direction** to `Property`, you then set the direction using the **Beamforming direction (deg)** parameter. When you select `Input port`, the direction is determined by the input to the `Ang` port.

Beamforming direction (deg) — Beamforming directions
2-by-*L* real-valued matrix

Beamforming directions, specified as a *2*-by-*L* real-valued matrix, where *L* is the number of beamforming directions. Each column takes the form `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this parameter, set the **Source of beamforming direction** parameter to `Property`.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Enable subband center frequencies output — Enable the output of subband center frequencies

off (default) | on

Select this check box to obtain the center frequencies of each subband via the output port, `Freq`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	Normal	Accelerator	Rapid Accelerator
Interpreted Execution	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
Code Generation	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use

Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or UCA.

Element spacing (m) — Spacing between array elements
0.5 for ULA arrays and [0.5,0.5] for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form [SpacingBetweenArrayRows,SpacingBetweenArrayColumns].
- When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to ULA or URA.

Array axis — Linear axis direction of ULA
y (default) | x | z

Linear axis direction of ULA, specified as y, x, or z. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

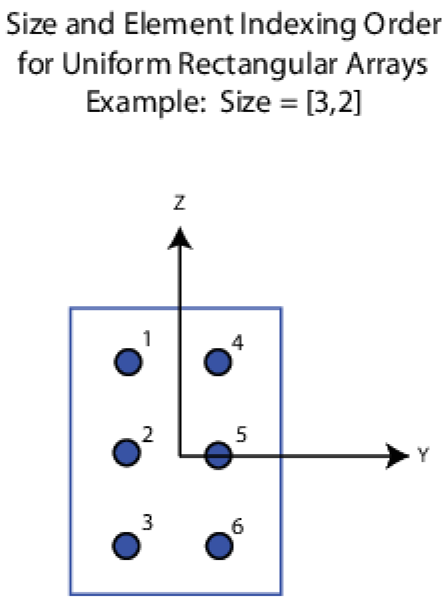
- To enable this parameter, set **Geometry** to ULA.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
[2,2] (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form [NumberOfArrayRows,NumberOfArrayColumns].
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of [3,2] creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to URA.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to URA.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.

Array Normal Parameter Value	Element Positions and Boresight Directions
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
[0;0;0] (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Element normals (deg) — Direction of conformal array element normal vectors
[0;0] | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

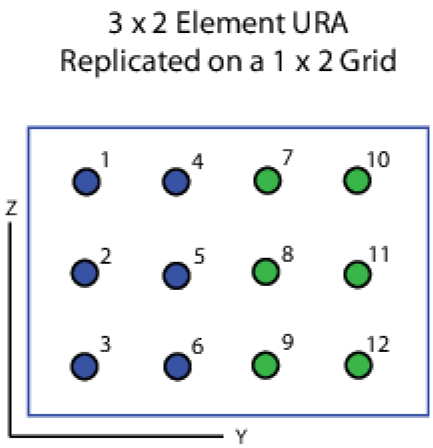
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as ULA or URA.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

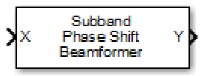
Introduced in R2015b

See Also

`phased.SubbandMVDRBeamformer`

Subband Phase Shift Beamformer

Subband phase shift beamformer



Libraries:
Phased Array System Toolbox / Beamforming

Description

The Subband Phase Shift Beamformer block performs delay-and-sum beamforming in the frequency domain. The signal is divided into frequency subbands. In each subband, a phase shift at the subband center frequency approximates the time delay. The resulting subband signals are summed to form the frequency-domain output signal and then converted to the time domain.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Ang — Beamforming direction
2-by-1 real-valued vector | 2-by-*L* real-valued matrix

Beamforming direction, specified as a 2-by-*L* real-valued matrix, where *L* is the number of beamforming directions. Each column takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Source of beamforming direction** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-*L* complex-valued matrix

Beamformed output, returned as an M-by-L complex-valued matrix. The quantity M is the number of signal samples and L is the number of desired beamforming directions specified by the `Beamforming direction` parameter or from the `Ang` port.

Data Types: `double`

Freq — Subband center frequencies
K-by-1 real-valued column vector

Subband center frequencies, returned as K-by-1 real-valued column vector. The quantity K is the number of subbands specified by the `Number of subbands` property.

Dependencies

To enable this port, select the **Enable subband center frequencies output** checkbox.

Data Types: `double`

W — Beamforming weights
N-by-L complex-valued matrix

Beamformed weights, returned as an N-by-L complex-valued matrix. The quantity N is the number of array elements. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, N represents the number of subarrays. L is the number of desired beamforming directions specified in the `Ang` port or by the `Beamforming direction (deg)` property. There is one set of weights for each beamforming direction.

Dependencies

To enable this port, select the **Enable weights output** checkbox.

Data Types: `double`

Parameters

Main tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Number of subbands — Number of processing subbands
`64` (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: `128`

Source of beamforming direction — Source of beamforming direction
`Property` (default) | `Input port`

Source of beamforming direction, specified as `Property` or `Input port`. When you set **Source of beamforming direction** to `Property`, you then set the direction using the **Beamforming direction (deg)** parameter. When you select `Input port`, the direction is determined by the input to the `Ang` port.

Beamforming direction (deg) — Beamforming directions
2-by-*L* real-valued matrix

Beamforming directions, specified as a *2*-by-*L* real-valued matrix, where *L* is the number of beamforming directions. Each column takes the form `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this parameter, set the **Source of beamforming direction** parameter to `Property`.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Enable subband center frequencies output — Enable the output of subband center frequencies

off (default) | on

Select this check box to obtain the center frequencies of each subband via the output port, `Freq`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `Partitioned array` | `Replicated subarray` | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `Partitioned array` — use the block parameters to specify the array.
- `Replicated subarray` — use the block parameters to specify the array.

- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

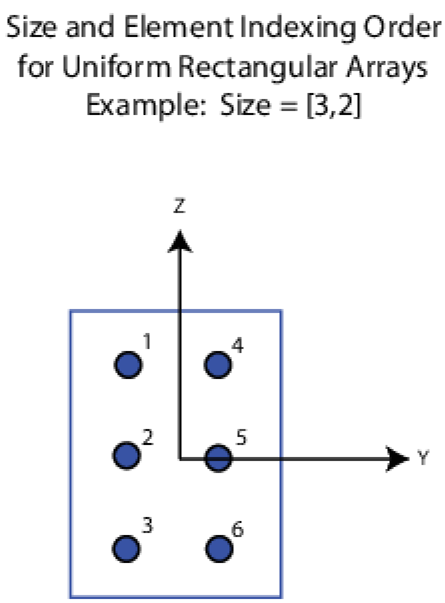
- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.
- When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

For a URA, array elements are indexed from top to bottom along the leftmost column, and then continue to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
x for URA arrays or z for UCA arrays (default) | y

Array normal direction, specified as x, y, or z.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
[0;0;0] (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position [x;y;z]of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

When you set **Specify sensor array as** to Replicated subarray, this parameter applies to each subarray.

Dependencies

To enable this parameter set **Geometry** to Conformal Array.

Element normals (deg) — Direction of conformal array element normal vectors
[0;0] | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. For a matrix, each column specifies the normal direction of the corresponding element in the form [azimuth;elevation] with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

Dependencies

To enable this parameter, set **Geometry** to `Conformal Array`.

Taper — Array element tapers
1 (default) | complex-valued scalar | complex-valued row vector

Element tapering, specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

When you set **Specify sensor array as** to `Replicated subarray`, this parameter applies to each subarray.

Subarray definition matrix — Define elements belonging to subarrays
logical matrix

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix represents a subarray and each entry in the row indicates when an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray lies at the subarray geometric center. The subarray geometric center depends on the **Subarray definition matrix** and **Geometry** parameters.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array`.

Subarray steering method — Specify subarray steering method
`None` (default) | `Phase` | `Time`

Subarray steering method, specified as one of

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Dependencies

To enable this parameter, set **Specify sensor array as** to `Partitioned array` or `Replicated subarray`.

Phase shifter frequency (Hz) — Subarray phase shifting frequency
`3.0e8` (default) | positive real-valued scalar

Operating frequency of subarray steering phase shifters, specified as a positive real-valued scalar. Units are Hz.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Number of bits in phase shifters — Subarray steering phase shift quantization bits
`0` (default) | non-negative integer

Subarray steering phase shift quantization bits, specified as a non-negative integer. A value of zero indicates that no quantization is performed.

Dependencies

To enable this parameter, set **Sensor array** to `Partitioned array` or `Replicated subarray` and set **Subarray steering method** to `Phase`.

Subarrays layout — Subarray position specification
`Rectangular` (default) | `Custom`

Specify the layout of replicated subarrays as `Rectangular` or `Custom`.

- When you set this parameter to `Rectangular`, use the **Grid size** and **Grid spacing** parameters to place the subarrays.
- When you set this parameter to `Custom`, use the **Subarray positions (m)** and **Subarray normals** parameters to place the subarrays.

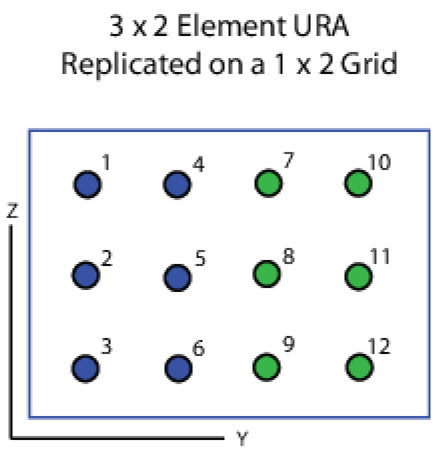
Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray`

Grid size — Dimensions of rectangular subarray grid
`[1,2]` (default)

Rectangular subarray grid size, specified as a single positive integer, or a 1-by-2 row vector of positive integers.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Grid spacing (m) — Spacing between subarrays on rectangular grid
`Auto` (default) | positive real-valued scalar | 1-by-2 vector of positive real-values

The rectangular grid spacing of subarrays, specified as a positive, real-valued scalar, a 1-by-2 row vector of positive, real-values, or `Auto`. Units are in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Subarray positions (m) — Positions of subarrays
`[0,0;0.5,0.5;0,0]` (default) | 3-by-*N* real-valued matrix

Positions of the subarrays in the custom grid, specified as a real 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array local coordinate system. The coordinates are expressed in the form `[x; y; z]`. Units are in meters.

Dependencies

To enable this parameter, set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Subarray normals — Direction of subarray normal vectors
`[0,0;0,0]` (default) | 2-by-*N* real matrix

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth;elevation]`. Angle units are in degrees. Angles are defined with respect to the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Dependencies

To enable this parameter, set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Version History

Introduced in R2014b

See Also

`phased.SubbandPhaseShiftBeamformer`

Time Delay Beamformer

Time-delay beamformer

Libraries:
Phased Array System Toolbox / Beamforming

Description

The Time Delay Beamformer block performs delay-and-sum beamforming. Plane-wave signals arriving at the array elements are time-aligned and then summed. Time alignment is achieved by transforming the signals into the frequency domain and applying linear phase shifts corresponding to a time delay. The individual signals are then added and converted back to the time domain.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Ang — Beamforming direction
2-by-1 real-valued vector

Beamforming direction, specified as a 2-by-1 real-valued vector. The vector takes the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Source of beamforming direction** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-1 complex-valued vector

Beamformed output, returned as an *M*-by-1 complex-valued vector. The quantity *M* is the number of signal samples.

Data Types: `double`

W — Beamforming weights
N-by-1 complex-valued vector

Beamformed weights, returned as an *N*-by-1 complex-valued vector. The quantity *N* is the number of array elements. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays.

Dependencies

To enable this port, select the **Enable weights output** checkbox.

Data Types: `double`

Parameters

Main tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Source of beamforming direction — Source of beamforming direction
`Property` (default) | `Input port`

Source of beamforming direction, specified as `Property` or `Input port`. When you set **Source of beamforming direction** to `Property`, you then set the direction using the **Beamforming**

direction (deg) parameter. When you select `Input port`, the direction is determined by the input to the `Ang` port.

Beamforming direction (deg) — Beamforming direction
2-by-1 real-valued vector

Beamforming direction, specified as a 2-by-1 real-valued vector taking the form `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this parameter, set the **Source of beamforming direction** parameter to `Property`.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><th rowspan="2">Block Simulation</th><th colspan="3">Simulation Behavior</th></tr>
<tr><th><code>Normal</code></th><th><code>Accelerator</code></th><th><code>Rapid Accelerator</code></th></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use

Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-Q vector. Q must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-P row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-P row vector. P must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.

- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-L row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-L vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-P row vector

Specify the polar pattern response angles, as a 1-by-P vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued L-by-P matrix

Specify the magnitude of the custom microphone element polar patterns as an L-by-P matrix. L is the number of frequencies specified in **Polar pattern frequencies (Hz)**. P is the number of angles

specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

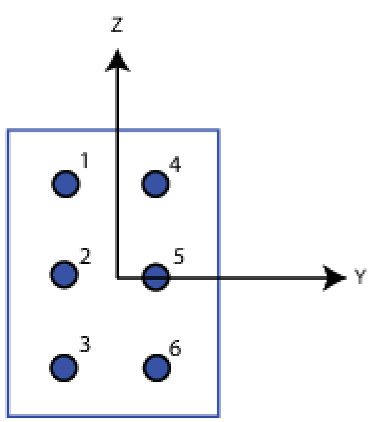
Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of elements in each row and column.

For a URA, array elements are indexed from top to bottom along the leftmost array column, and continued to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to URA.

Array normal — Array normal direction
x for URA arrays or z for UCA arrays (default) | y

Array normal direction, specified as x, y, or z.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to URA or UCA.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to UCA.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Data Types: `double`

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. If the parameter value is a matrix, each

column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

To enable this parameter, set **Geometry** to `Conformal Array`.

Data Types: `double`

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Version History

Introduced in R2014b

See Also

`phased.TimeDelayBeamformer`

Time Delay LCMV Beamformer

Time delay LCMV beamformer

Libraries:
Phased Array System Toolbox / Beamforming

Description

The Time Delay LCMV Beamformer block performs time-delay linear constraint minimum variance (LCMV) beamforming.

Ports

Input

X — Input signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

XT — Training signal
M-by-*N* complex-valued matrix

Input signal, specified as an *M*-by-*N* matrix, where *M* is the number of samples in the data, and *N* is the number of array elements.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Dependencies

To enable this port, select the **Enable training data input** check box.

Data Types: `double`

Ang — Beamforming direction
2-by-1 real-valued vector

Beamforming direction, specified as a 2-by-1 real-valued matrix taking the form of `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°, inclusive, and the elevation angle must lie between –90° and 90°, inclusive. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this port, set the **Source of beamforming direction** parameter to `Input port`.

Data Types: `double`

Output

Y — Beamformed output
M-by-1 complex-valued vector

Beamformed output, returned as an *M*-by-1 complex-valued vector. The quantity *M* is the number of signal samples.

Data Types: `double`

W — Beamforming weights
N-by-1 complex-valued vector

Beamformed weights, returned as an *N*-by-1 complex-valued vector. The quantity *N* is the number of array elements. When the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`, *N* represents the number of subarrays.

Dependencies

To enable this port, select the **Enable weights output** checkbox.

Data Types: `double`

Parameters

Main tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

FIR filter length — FIR filter length
`1` (default) | positive integer

The length of the FIR filter used to process each sensor element data, specified as a positive integer.

Data Types: `double`

Constraint matrix — Constraint matrix used for time-delay LCMV beamformer
`[1;1]` | complex-valued M-by-K matrix

The constraint matrix used for time-delay LCMV beamformer, specified as a complex-valued$M$-by-K matrix. Each column of the matrix is a constraint and M is the degrees of freedom of the beamformer. For a time delay LCMV beamformer, M is given by the product of the number of elements of the array and the value of the **FIR filter length** parameter.

Data Types: `double`

Desired response vector — Desired response of time-delay LCMV beamformer
`1` (default) | K column vector

Desired response used for time-delay LCMV beamformer, specified as a length-K column vector. K is the number of constraints in the **Constraint matrix** parameter. Each element in the vector defines the desired response of the constraint specified in the corresponding column of the **Constraint matrix** parameter matrix.

Diagonal loading factor — Diagonal loading factor for stability
nonnegative scalar

Specify the diagonal loading factor as a nonnegative scalar. Diagonal loading is a technique used to achieve robust beamforming performance, especially when the sample support is small.

Enable training data input — Enable the use of training data
off (default) | on

Select this check box to specify additional training data via the input port XT. To use the input signal as the training data, clear the check box which removes the port.

Source of beamforming direction — Source of beamforming direction
`Property` (default) | `Input port`

Source of beamforming direction, specified as `Property` or `Input port`. When you set **Source of beamforming direction** to `Property`, you then set the direction using the **Beamforming direction (deg)** parameter. When you select `Input port`, the direction is determined by the input to the `Ang` port.

Beamforming direction (deg) — Beamforming direction
2-by-1 real-valued vector

Beamforming direction, specified as a 2-by-1 real-valued vector taking the form `[AzimuthAngle;ElevationAngle]`. Angle units are in degrees. The azimuth angle must lie between –180° and 180°. The elevation angle must lie between –90° and 90°. Angles are defined with respect to the local coordinate system of the array.

Dependencies

To enable this parameter, set the **Source of beamforming direction** parameter to `Property`.

Enable weights output — Option to output beamformer weights
off (default) | on

Select this check box to obtain the beamformer weights from the output port, `W`.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`

Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Arrays Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-P row vector. P must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-Q vector. Q must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-P row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-P row vector. P must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.

- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles

specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Geometry — Array geometry
`ULA` (default) | `URA` | `UCA` | `Conformal Array`

Array geometry, specified as one of

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements — Number of array elements
`2` for ULA arrays and `5` for UCA arrays (default) | integer greater than or equal to 2

The number of array elements for ULA or UCA arrays, specified as an integer greater than or equal to 2.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `UCA`.

Element spacing (m) — Spacing between array elements
`0.5` for ULA arrays and `[0.5,0.5]` for URA arrays (default) | positive scalar for ULA or URA arrays | 2-element vector of positive values for URA arrays

Spacing between adjacent array elements:

- ULA — specify the spacing between two adjacent elements in the array as a positive scalar.
- URA — specify the spacing as a positive scalar or a 1-by-2 vector of positive values. If **Element spacing (m)** is a scalar, the row and column spacings are equal. If **Element spacing (m)** is a vector, the vector has the form `[SpacingBetweenArrayRows,SpacingBetweenArrayColumns]`.

Dependencies

To enable this parameter, set **Geometry** to `ULA` or `URA`.

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Dependencies

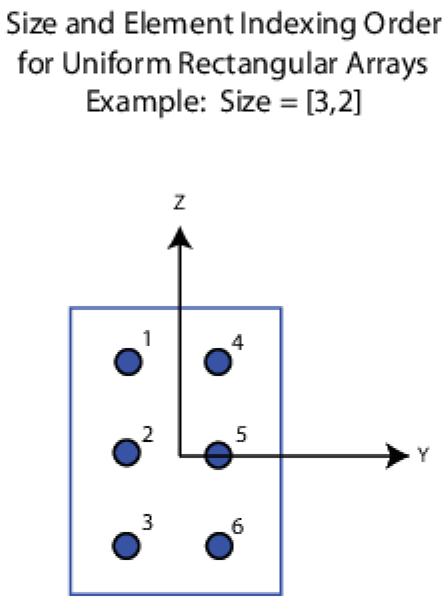
- To enable this parameter, set **Geometry** to `ULA`.
- This parameter is also enabled when the block only supports ULA arrays.

Array size — Dimensions of URA array
`[2,2]` (default) | positive integer | 1-by-2 vector of positive integers

Dimensions of a URA array, specified as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of elements in each row and column.

For a URA, array elements are indexed from top to bottom along the leftmost array column, and continued to the next columns from left to right. In this figure, the **Array size** value of `[3,2]` creates an array having three rows and two columns.



Dependencies

To enable this parameter, set **Geometry** to `URA`.

Element lattice — Lattice of URA element positions
`Rectangular` (default) | `Triangular`

Lattice of URA element positions, specified as `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive row-axis direction. The displacement is one-half the element spacing along the row dimension.

Dependencies

To enable this parameter, set **Geometry** to `URA`.

Array normal — Array normal direction
`x` for URA arrays or `z` for UCA arrays (default) | `y`

Array normal direction, specified as `x`, `y`, or `z`.

Elements of planar arrays lie in a plane orthogonal to the selected array normal direction. Element boresight directions point along the array normal direction.

Array Normal Parameter Value	Element Positions and Boresight Directions
x	Array elements lie in the *yz*-plane. All element boresight vectors point along the *x*-axis.
y	Array elements lie in the *zx*-plane. All element boresight vectors point along the *y*-axis.
z	Array elements lie in the *xy*-plane. All element boresight vectors point along the *z*-axis.

Dependencies

To enable this parameter, set **Geometry** to `URA` or `UCA`.

Radius of UCA (m) — UCA array radius
0.5 (default) | positive scalar

Radius of UCA array, specified as a positive scalar.

Dependencies

To enable this parameter, set **Geometry** to `UCA`.

Element positions (m) — Positions of conformal array elements
`[0;0;0]` (default) | 3-by-*N*matrix of real values

Positions of the elements in a conformal array, specified as a 3-by-*N* matrix of real values, where *N* is the number of elements in the conformal array. Each column of this matrix represents the position `[x;y;z]`of an array element in the array local coordinate system. The origin of the local coordinate system is *(0,0,0)*. Units are in meters.

Dependencies

To enable this parameter set **Geometry** to `Conformal Array`.

Data Types: `double`

Element normals (deg) — Direction of conformal array element normal vectors
`[0;0]` | 2-by-1 column vector | 2-by-*N* matrix

Direction of element normal vectors in a conformal array, specified as a 2-by-1 column vector or a 2-by-*N* matrix. *N* indicates the number of elements in the array. If the parameter value is a matrix, each

column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If the parameter value is a 2-by-1 column vector, the same pointing direction is used for all array elements.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal direction.

To enable this parameter, set **Geometry** to `Conformal Array`.

Data Types: `double`

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Version History

Introduced in R2014b

See Also

`phased.TimeDelayLCMVBeamformer`

Time Varying Gain

Time varying gain (TVG) control

Libraries:
Phased Array System Toolbox / Detection

Description

The Time Varying Gain block applies a time-varying gain to input signals to compensate for geometric range loss at each range gate. Time varying gain (TVG) is sometimes called automatic gain control (AGC).

Ports

Input

Port_1 — Input signal
vector | matrix

Input signal, specified as a vector or matrix.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double` | `single`

Port_2 — Range loss
column vector

Range loss, specified as a column vector. The length of p must be equal to or greater than the number of rows of X.

The process equalizes power levels across all samples to match a given reference range. The compensated signal is returned in Y. X can be a column vector, a matrix, or a cube. The gain is applied to each column in X independently. The number of rows in X cannot exceed the length of the loss vector specified in the `RangeLoss` property. Y has the same dimensionality as X. X can be single or double precision.

Dependencies

To enable this port, set **Source of range losses** parameter to `Input port`.

Data Types: `double` | `single`

Output

Port_1 — Range-compensated input signal
vector | matrix

Range-compensated Input signal, returned as a vector or matrix. The output has the same size as the input signal.

Data Types: `double` | `single`

Parameters

Source of range losses — Source of range losses

`Property` (default) | `Input Port`

Range loss source, specified as `Property` or `Input Port`.

`Property`	Range losses are specified by the **Range loss (dB)** parameter.
`Input port`	Range losses are specified using the input port `L` .

Range losses (dB) — Loss at each input sample range

`0` (default)

Loss at each input sample range, specified as a vector — elements correspond to the samples in the input signal. Units are in dB.

Reference range loss (dB) — Loss at reference range

`0` (default)

Specify the loss at a given reference range as a scalar. Units are in dB.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><td rowspan="2">Block Simulation</td><td colspan="3">Simulation Behavior</td></tr>
<tr><td><code>Normal</code></td><td><code>Accelerator</code></td><td><code>Rapid Accelerator</code></td></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use

Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History

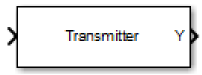
Introduced in R2014b

See Also

`phased.TimeVaryingGain`

Transmitter

Amplify and transmit a signal



Libraries:
Phased Array System Toolbox / Transmitters and Receivers

Description

The Transmitter block amplifies and transmits waveform pulses. The transmitter can either maintain coherence between pulses or insert phase noise.

Ports

Input

Input 1 — Input signal
complex-valued vector

Input signal, specified as a complex-valued vector.

Data Types: `double`

Output

Y — Transmitted signal
complex-valued vector

Transmitted signal, returned as a complex-valued vector. The transmitted signal is the amplified input signal **Input 1** where the amplification is based on the characteristics of the transmitter, such as the peak power and the gain.

Data Types: `double`

TR — Transmitter status
logical vector

Transmitter status, output as a logical vector where `true` indicates the transmitter is on for the corresponding sample time, and `false` indicates the transmitter is off.

Dependencies

To enable this port, select the **Enable transmitter status output** check box.

Data Types: `double`

Ph — Output phase noise
vector

Added phase noise, output as a vector. Random phase noise added to each transmitted sample. `PHNOISE` is a vector which has the same dimension as Y. Each element in `PHNOISE` contains the random phase between 0 and 2*pi, added to the corresponding sample in `Y` by the transmitter.

Dependencies

To enable this port, select the **Enable pulse phase noise output** check box and deselect the **Preserve coherence among pulses** check box.

Data Types: `double`

Parameters

Peak power (W) — Peak power

`5000` (default) | positive scalar

Transmitted peak power, specified as a positive scalar. Units are in Watts.

Gain (dB) — Transmitter gain

`20` (default) | scalar

Transmit gain, specified as a scalar. Units are in dB.

Loss factor (dB) — Loss factor

`0` (default)

Transmit loss factor, specified as a nonnegative scalar. Units are in dB.

Enable transmitter status output — Enable transmitter status output

off (default) | on

Select this check box to send the transmitter-in-use status for each output sample from the output port TR. From the output port, a 1 indicates that the transmitter is on, and a 0 indicates that the transmitter is off.

Preserve coherence among pulses — Preserve coherence among pulses

on (default) | off

Select this check box to preserve coherence among transmitted pulses. When you select this box, the transmitter does not introduce any random phases to the output pulses. When you clear this box, the transmitter adds a random phase noise to each transmitted pulse. The random phase noise is introduced by multiplying the pulse value by $e^{j\phi}$where ϕ is a uniform random variable on the interval $[0,2\pi]$.

Enable pulse phase noise output — Enable pulse phase noise output

off (default) | on

Select this check box to create an output port, Ph, with the output sample's random phase noise introduced if **Preserve coherence among pulses** is cleared. The output port can be directed to a receiver to simulate coherent-on-receive systems.

Dependencies

This check box appears only when **Preserve coherence among pulses** is deselected.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

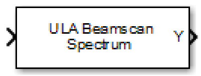
Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2014b

See Also
`phased.Transmitter`

ULA Beamscan Spectrum

Beamscan spatial spectrum estimator for ULA



Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The ULA Beamscan Spectrum block estimates the spatial spectrum of incoming narrowband signals by scanning a region of broadside angles using a narrowband conventional beamformer applied to a uniform linear array. The block optionally calculates the direction of arrival of a specified number of signals by estimating peaks of the spectrum.

Ports

Input

Port 1 — Received signal
M-by-*N* complex-valued matrix

Received signal, specified as an *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values (snapshots) contained in the signal and *N* is the number of sensor elements in the array.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Output

Y — Beamscan spatial spectrum
non-negative, real-valued column vector

Beamscan spatial spectrum, returned as a non-negative, real-valued column vector representing the magnitude of the estimated beamscan spatial spectrum. Each entry corresponds to an angle specified by the **Scan angles (deg)** parameter.

Data Types: `double`

Ang — Directions of arrival
non-negative, real-valued column vector

Directions of arrival of the signals, returned as a real-valued row vector. The direction of arrival angle is the broadside angle between the source direction and the array axis. Angle units are in degrees. The length of the vector is the number of signals specified by the **Number of signals** parameter. If the object cannot identify peaks in the spectrum, it will return `NaN`.

Dependencies

To enable this output port, select the **Enable DOA output** check box.

Data Types: `double`

Parameters

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Number of bits in phase shifters — Number of phase shift quantization bits
`0` (default) | nonnegative integer

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Forward-backward averaging — Enable forward-backward averaging
off (default) | on

Select this parameter to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold structure.

Spatial smoothing — Enable spatial smoothing
`0` (default) | non-negative integer

Specify the amount of averaging used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each increase in smoothing handles one extra coherent source, but reduces the effective number of elements by one. The maximum value of this parameter is $N - 2$, where N is the number of sensors in the ULA.

Scan angles (deg) — Search angles for spectrum peaks
`-90:90` (default) | real-valued row vector

Specify the scan angles in degrees as a real-valued row vector. The angles are array broadside angles and must lie between –90° and 90°, inclusive. You must specify the angles in increasing order.

Enable DOA output — Output directions of arrival through output port
off (default) | on

Select this parameter to output the signals directions of arrival (DOA) through the **Ang** output port.

Number of signals — Expected number of arriving signals
1 (default) | positive integer

Specify the expected number of signals for DOA estimation as a positive scalar integer.

Dependencies

To enable this parameter, select the **Enable DOA output** check box.

Data Types: `double`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum

Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.
- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Radiation pattern (dB) — Custom antenna radiation pattern
`zeros(181,361)` | complex-valued matrix | complex-valued MATLAB array

Magnitude of the combined polarized antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array. The value of *Q* must equal the value of *Q* specified by **Elevation angles (deg)**. The value of *P* must equal the value of *P* specified by **Azimuth angles (deg)**. The value of *L* must equal the value of *L* specified by **Operating frequency vector (Hz)**.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Number of elements — Number of array elements in U
2 (default) | positive integer greater than or equal to two

The number of array elements for ULA arrays, specified as an integer greater than or equal to two.

Example: `11`

Data Types: `double`

Element spacing — Distance between ULA elements
`0.5` (default) | positive scalar

Distance between adjacent ULA elements, specified as a positive scalar. Units are in meters.

Example: `1.5`

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. Then, all ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Version History

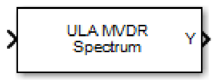
Introduced in R2014b

See Also

`phased.BeamscanEstimator`

ULA MVDR Spectrum

MVDR spatial spectrum estimator for ULA



Library

Direction of Arrival (DOA)

`phaseddoalib`

Description

The ULA MVDR Spectrum block estimates the spatial spectrum of incoming narrowband signals by scanning a region of broadside angles using a narrowband minimum variance distortionless response (MVDR) beamformer for a uniform linear array. The block optionally calculates the direction of arrival (DOA) of a specified number of signals by estimating peaks of the spectrum. The MVDR DOA estimator is also called the Capon DOA estimator.

Parameters

Signal Propagation speed (m/s)

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can use the function `physconst` to specify the speed of light.

Operating frequency (Hz)

Specify the operating frequency of the system, in hertz, as a positive scalar.

Number of bits in phase shifters

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Forward-backward averaging

Select this parameter to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold.

Spatial smoothing

Specify the amount of averaging, L, used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each increase in smoothing handles one extra coherent source, but reduces the effective number of elements by one. The maximum value of this parameter is $N – 2$, where N is the number of sensors.

Scan angles (deg)

Specify the scan angles in degrees as a real vector. The angles are broadside angles and must be between –90° and 90°, inclusive. You must specify the angles in increasing order.

Enable DOA output

Select this parameter to output the signals directions of arrival (DOA) through the **Ang** output port. Selecting this parameter enables the **Number of signals** parameter.

Number of signals

Specify the number of signals for DOA estimation as a positive scalar integer. This parameter appears when you select the **Enable DOA output** check box.

Simulate using

Block simulation method, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster than they would in interpreted execution. You can run repeated executions without recompiling. However, if you change any block parameters, then the block automatically recompiles before execution.

When setting this parameter, you must take into account the overall model simulation mode. The table shows how the **Simulate using** parameter interacts with the overall simulation mode.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Array Parameters

Specify sensor array as

Specify a ULA sensor array directly or by using a MATLAB expression.

Types

`Array (no subarrays)`
`MATLAB expression`

Number of elements

Specifies the number of elements in the array as an integer.

Element spacing

Specify the spacing, in meters, between two adjacent elements in the array.

Array axis

This parameter appears when the **Geometry** parameter is set to `ULA` or when the block only supports a ULA array geometry. Specify the array axis as `x`, `y`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Taper

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Expression

A valid MATLAB expression containing a constructor for a uniform linear array, for example, `phased.ULA`.

Sensor Array Tab: Element Parameters

Element type

Specify antenna or microphone type as

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Exponent of cosine pattern

This parameter appears when you set **Element type** to `Cosine Antenna`.

Specify the exponent of the cosine pattern as a scalar or a 1-by-2 vector. You must specify all values as non-negative real numbers. When you set **Exponent of cosine pattern** to a scalar, both the azimuth direction cosine pattern and the elevation direction cosine pattern are raised to the specified value. When you set **Exponent of cosine pattern** to a 1-by-2 vector, the first element is the exponent for the azimuth direction cosine pattern and the second element is the exponent for the elevation direction cosine pattern.

Operating frequency range (Hz)

This parameter appears when **Element type** is set to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Specify the operating frequency range, in hertz, of the antenna element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The antenna element has no response outside the specified frequency range.

Operating frequency vector (Hz)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify the frequencies, in Hz, at which to set the antenna and microphone frequency responses as a 1-by-*L* row vector of increasing values. Use **Frequency responses** to set the frequency responses. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of **Operating frequency vector (Hz)**.

Frequency responses (dB)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify this parameter as the frequency response of an antenna or microphone, in decibels, for the frequencies defined by **Operating frequency vector (Hz)**. Specify **Frequency responses (dB)** as a 1-by-*L* vector matching the dimensions of the vector specified in **Operating frequency vector (Hz)**.

Input Pattern Coordinate System

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Azimuth angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Azimuth angles must lie between –180° and 180° and be in strictly increasing order.

Elevation angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90° and be in strictly increasing order.

Phi Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Phi angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Theta Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Theta angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 180° and be in strictly increasing order.

Magnitude pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Phase pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

MatchArrayNormal

This parameter appears when the **Element type** is set to `Custom Antenna`.

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Polar pattern frequencies (Hz)

This parameter appears when the **Element type** is set to `Custom Microphone`.

Specify the measuring frequencies of the polar patterns as a 1-by-*M* vector. The measuring frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** parameter. Frequency units are in Hz.

Polar pattern angles (deg)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the measuring angles of the polar patterns, as a 1-by-*N* vector. The angles are measured from the central pickup axis of the microphone, and must be between –180° and 180°, inclusive.

Polar pattern (dB)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the magnitude of the microphone element polar pattern as an *M*-by-*N* matrix. *M* is the number of measuring frequencies specified in **Polar pattern frequencies (Hz)**. *N* is the number of measuring angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. Assume that the pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. Assume that the polar pattern is symmetric around the central axis. You can construct the microphone's response pattern in 3-D space from the polar pattern.

Baffle the back of the element

This check box appears only when the **Element type** parameter is set to `Isotropic Antenna` or `Omni Microphone`.

Select this check box to baffle the back of the antenna element. In this case, the antenna responses to all azimuth angles beyond ±90° from broadside are set to zero. Define the broadside direction as 0° azimuth angle and 0° elevation angle.

Ports

Note The block input and output ports correspond to the input and output parameters described in the `step` method of the underlying System object. See link at the bottom of this page.

Port	Description	Supported Data Types
`In`	Input signal. The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.	Double-precision floating point
`Ang`	Estimated broadside DOA angles.	Double-precision floating point
`Y`	Estimated spatial spectrum.	Double-precision floating point

Version History

Introduced in R2014b

See Also

`phased.MVDREstimator`

ULA MUSIC Spectrum

MUSIC spatial spectrum estimator for ULA

Libraries:
Phased Array System Toolbox / Direction of Arrival

Description

The ULA MUSIC Spectrum block estimates the spatial spectrum of incoming narrowband signals using the MUSIC algorithm. The algorithm computes the MUSIC pseudo-spectrum of a ULA by scanning a region of broadside angles. The block optionally calculates the direction of arrival (DOA) of a specified number of signals by estimating peaks of the spectrum.

Ports

Input

Port 1 — Received signal
M-by-*N* complex-valued matrix

Received signal, specified as an *M*-by-*N* complex-valued matrix. The quantity *M* is the number of sample values (snapshots) contained in the signal and *N* is the number of sensor elements in the array.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Output

Y — MUSIC spatial spectrum
non-negative, real-valued column vector

MUSIC spatial spectrum, returned as a non-negative, real-valued column vector representing the magnitude of the estimated MUSIC spatial spectrum. Each entry corresponds to an angle specified by the **Scan angles (deg)** parameter.

Data Types: `double`

Ang — Directions of arrival
non-negative, real-valued column vector

Directions of arrival of the signals, returned as a real-valued row vector. The direction of arrival angle is the broadside angle between the source direction and the array axis. The length of the vector is the number of signals specified by the `Number of signals` parameter. If the object cannot identify peaks in the spectrum, it will return `NaN`. Angle units are in degrees.

Dependencies

Select the **Enable DOA output** parameter to enable this output port.

Data Types: `double`

Parameters

Main Tab

Signal propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | real-valued positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`. Units are in meters per second.

Example: `3e8`

Data Types: `double`

Operating frequency (Hz) — System operating frequency
`3e8` (default) | positive real scalar

System operating frequency, specified as a positive scalar. Units are in Hz.

Forward-backward averaging — Enable forward-backward averaging
off (default) | on

Select this parameter to use forward-backward averaging to estimate the covariance matrix for sensor arrays with a conjugate symmetric array manifold structure.

Spatial smoothing — Enable spatial smoothing
`0` (default) | non-negative integer

Specify the amount of averaging used by spatial smoothing to estimate the covariance matrix as a nonnegative integer. Each increase in smoothing handles one extra coherent source, but reduces the effective number of elements by one. The maximum value of this parameter is $N - 2$, where N is the number of sensors in the ULA.

Scan angles (deg) — Search angles for spectrum peaks
`-90:90` (default) | real-valued row vector

Specify the scan angles in degrees as a real-valued row vector. The angles are array broadside angles and must lie between –90° and 90°, inclusive. You must specify the angles in increasing order.

Enable DOA output — Output directions of arrival through output port
off (default) | on

Select this parameter to output the signals directions of arrival (DOA) through the **Ang** output port.

Number of signals — Expected number of arriving signals
1 (default) | positive integer

Specify the expected number of signals for DOA estimation as a positive scalar integer.

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Sensor Array Tab

Specify sensor array as — Method to specify array
`Array (no subarrays)` (default) | `MATLAB expression`

Method to specify array, specified as `Array (no subarrays)` or `MATLAB expression`.

- `Array (no subarrays)` — use the block parameters to specify the array.

- `MATLAB expression` — create the array using a MATLAB expression.

Expression — MATLAB expression used to create an array
Phased Array System Toolbox array System object

MATLAB expression used to create an array, specified as a valid Phased Array System Toolbox array System object.

Example: `phased.URA('Size',[5,3])`

Dependencies

To enable this parameter, set **Specify sensor array as** to `MATLAB expression`.

Element Parameters

Element type — Array element types
`Isotropic Antenna` (default) | `Cosine Antenna` | `Custom Antenna` | `Omni Microphone` | `Custom Microphone`

Antenna or microphone type, specified as one of the following:

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Operating frequency range (Hz) — Operating frequency range of the antenna or microphone element
`[0,1e20]` (default) | real-valued 1-by-2 row vector

Specify the operating frequency range of the antenna or microphone element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The element has no response outside this frequency range. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Operating frequency vector (Hz) — Operating frequency range of custom antenna or microphone elements
`[0,1e20]` (default) | real-valued row vector

Specify the frequencies at which to set antenna and microphone frequency responses as a 1-by-*L* row vector of increasing real values. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of this vector. Frequency units are in Hz.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`. Use **Frequency responses (dB)** to set the responses at these frequencies.

Baffle the back of the element — Set back response of an `Isotropic Antenna` element or an `Omni Microphone` element to zero
off (default) | on

Select this check box to baffle the back response of the element. When back baffled, the responses at all azimuth angles beyond ±90° from broadside are set to zero. The broadside direction is defined as 0° azimuth angle and 0° elevation angle.

Dependencies

To enable this check box, set **Element type** to `Isotropic Antenna` or `Omni Microphone`.

Exponent of cosine pattern — Exponents of azimuth and elevation cosine patterns
`[1.5 1.5]` (default) | nonnegative scalar | real-valued 1-by-2 matrix of nonnegative values

Specify the exponents of the cosine pattern as a nonnegative scalar or a real-valued 1-by-2 matrix of nonnegative values. When **Exponent of cosine pattern** is a 1-by-2 vector, the first element is the exponent in the azimuth direction and the second element is the exponent in the elevation direction. When you set this parameter to a scalar, both the azimuth direction and elevation direction cosine patterns are raised to the same power.

Dependencies

To enable this parameter, set **Element type** to `Cosine Antenna`.

Frequency responses (dB) — Antenna and microphone frequency response
`[0,0]` (default) | real-valued row vector

Frequency response of a custom antenna or custom microphone for the frequencies defined by the **Operating frequency vector (Hz)** parameter. The dimensions of **Frequency responses (dB)** must match the dimensions of the vector specified by the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna` or `Custom Microphone`.

Input Pattern Coordinate System — Coordinate system of custom antenna pattern
`az-el` (default) | `phi-theta`

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Azimuth angles (deg) — Azimuth angles of antenna radiation pattern
`[-180:180]` (default) | real-valued row vector

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Azimuth angles must lie between –180° and 180°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Elevation angles (deg) — Elevation angles of antenna radiation pattern
`[-90:90]` (default) | real-valued row vector

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90°, inclusive, and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `az-el`.

Phi Angles (deg) — Phi angle coordinates of custom antenna radiation pattern
`0:360` | real-valued 1-by-*P* row vector

Phi angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Theta Angles (deg) — Theta angle coordinates of custom antenna radiation pattern
`0:180` | real-valued 1-by-*Q* row vector

Theta angles of points at which to specify the antenna radiation pattern, specify as a real-valued 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 360° and be in strictly increasing order.

Dependencies

To enable this parameter, set the **Element type** parameter to `Custom Antenna` and the **Input Pattern Coordinate System** parameter to `phi-theta`.

Radiation pattern (dB) — Custom antenna radiation pattern
`zeros(181,361)` | complex-valued matrix | complex-valued MATLAB array

Magnitude of the combined polarized antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array. The value of *Q* must equal the value of *Q* specified by **Elevation angles (deg)**. The value of *P* must equal the value of *P* specified by **Azimuth angles (deg)**. The value of *L* must equal the value of *L* specified by **Operating frequency vector (Hz)**.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Magnitude pattern (dB) — Magnitude of combined antenna radiation pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Phase pattern (deg) — Custom antenna radiation phase pattern
`zeros(181,361)` (default) | real-valued *Q*-by-*P* matrix | real-valued *Q*-by-*P*-by-*L* array

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (**

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

MatchArrayNormal — Rotate antenna element to array normal
`on` (default) | `off`

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Dependencies

To enable this parameter, set **Element type** to `Custom Antenna`.

Polar pattern frequencies (Hz) — Polar pattern microphone response frequencies
1e3 (default) | real scalar | real-valued 1-by-*L* row vector

Polar pattern microphone response frequencies, specified as a real scalar, or a real-valued, 1-by-*L* vector. The response frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** vector.

Dependencies

To enable this parameter, set **Element type** set to `Custom Microphone`.

Polar pattern angles (deg) — Polar pattern response angles
`[-180:180]` (default) | real-valued -by-*P* row vector

Specify the polar pattern response angles, as a 1-by-*P* vector. The angles are measured from the central pickup axis of the microphone and must be between –180° and 180°, inclusive.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Polar pattern (dB) — Custom microphone polar response
`zeros(1,361)` (default) | real-valued *L*-by-*P* matrix

Specify the magnitude of the custom microphone element polar patterns as an *L*-by-*P* matrix. *L* is the number of frequencies specified in **Polar pattern frequencies (Hz)**. *P* is the number of angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. The pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees

azimuth and 0° degrees elevation. The polar pattern is symmetric around the central axis. You can construct the microphone response pattern in 3-D space from the polar pattern.

Dependencies

To enable this parameter, set **Element type** to `Custom Microphone`.

Array Parameters

Number of elements — Number of array elements in U
2 (default) | positive integer greater than or equal to two

The number of array elements for ULA arrays, specified as an integer greater than or equal to two.

Example: `11`

Data Types: `double`

Element spacing — Distance between ULA elements
`0.5` (default) | positive scalar

Distance between adjacent ULA elements, specified as a positive scalar. Units are in meters.

Example: `1.5`

Array axis — Linear axis direction of ULA
`y` (default) | `x` | `z`

Linear axis direction of ULA, specified as `y`, `x`, or `z`. Then, all ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Taper — Array element tapers
1 (default) | complex scalar | complex-valued row vector

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array.

Also known as element weights, tapers multiply the array element responses. Tapers modify both amplitude and phase of the response to reduce side lobes or steer the main response axis.

If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. The number of weights must match the number of elements of the array.

Data Types: `double`

Version History

Introduced in R2016b

See Also

Blocks
MUSIC Spectrum

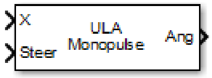
Objects
`phased.MUSICEstimator` | `phased.ULA`

Functions
`musicdoa`

Topics
"MUSIC Super-Resolution DOA Estimation"

ULA Sum and Difference Monopulse

Sum-and-difference monopulse tracker for ULA



Library

Direction of Arrival (DOA)

`phaseddoalib`

Description

The ULA Sum-and-Difference Monopulse block estimates the direction of arrival of a narrowband signal on a uniform linear array based on an initial guess using a sum-and-difference monopulse algorithm. The block obtains the difference steering vector by phase-reversing the latter half of the sum steering vector.

Parameters

Signal Propagation speed (m/s)

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can use the function `physconst` to specify the speed of light.

Operating frequency (Hz)

Specify the operating frequency of the system, in hertz, as a positive scalar.

Number of bits in phase shifters

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Simulate using

Block simulation method, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster than they would in interpreted execution. You can run repeated executions without recompiling. However, if you change any block parameters, then the block automatically recompiles before execution.

When setting this parameter, you must take into account the overall model simulation mode. The table shows how the **Simulate using** parameter interacts with the overall simulation mode.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Array Parameters

Specify sensor array as

Specify a ULA sensor array directly or by using a MATLAB expression.

Types

`Array (no subarrays)`
`MATLAB expression`

Number of elements

Specifies the number of elements in the array as an integer.

Element spacing

Specify the spacing, in meters, between two adjacent elements in the array.

Array axis

This parameter appears when the **Geometry** parameter is set to `ULA` or when the block only supports a ULA array geometry. Specify the array axis as `x`, `y`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Taper

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Expression

A valid MATLAB expression containing a constructor for a uniform linear array, for example, `phased.ULA`.

Sensor Array Tab: Element Parameters

Element type

Specify antenna or microphone type as

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Exponent of cosine pattern

This parameter appears when you set **Element type** to `Cosine Antenna`.

Specify the exponent of the cosine pattern as a scalar or a 1-by-2 vector. You must specify all values as non-negative real numbers. When you set **Exponent of cosine pattern** to a scalar, both the azimuth direction cosine pattern and the elevation direction cosine pattern are raised to the specified value. When you set **Exponent of cosine pattern** to a 1-by-2 vector, the first element is the exponent for the azimuth direction cosine pattern and the second element is the exponent for the elevation direction cosine pattern.

Operating frequency range (Hz)

This parameter appears when **Element type** is set to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Specify the operating frequency range, in hertz, of the antenna element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The antenna element has no response outside the specified frequency range.

Operating frequency vector (Hz)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify the frequencies, in Hz, at which to set the antenna and microphone frequency responses as a 1-by-*L* row vector of increasing values. Use **Frequency responses** to set the frequency responses. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of **Operating frequency vector (Hz)**.

Frequency responses (dB)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify this parameter as the frequency response of an antenna or microphone, in decibels, for the frequencies defined by **Operating frequency vector (Hz)**. Specify **Frequency responses (dB)** as a 1-by-*L* vector matching the dimensions of the vector specified in **Operating frequency vector (Hz)**.

Input Pattern Coordinate System

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Azimuth angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Azimuth angles must lie between –180° and 180° and be in strictly increasing order.

Elevation angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90° and be in strictly increasing order.

Phi Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Phi angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Theta Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Theta angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 180° and be in strictly increasing order.

Magnitude pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Phase pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

MatchArrayNormal

This parameter appears when the **Element type** is set to `Custom Antenna`.

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Polar pattern frequencies (Hz)

This parameter appears when the **Element type** is set to `Custom Microphone`.

Specify the measuring frequencies of the polar patterns as a 1-by-*M* vector. The measuring frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** parameter. Frequency units are in Hz.

Polar pattern angles (deg)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the measuring angles of the polar patterns, as a 1-by-*N* vector. The angles are measured from the central pickup axis of the microphone, and must be between –180° and 180°, inclusive.

Polar pattern (dB)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the magnitude of the microphone element polar pattern as an *M*-by-*N* matrix. *M* is the number of measuring frequencies specified in **Polar pattern frequencies (Hz)**. *N* is the number of measuring angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. Assume that the pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. Assume that the polar pattern is symmetric around the central axis. You can construct the microphone's response pattern in 3-D space from the polar pattern.

Baffle the back of the element

This check box appears only when the **Element type** parameter is set to `Isotropic Antenna` or `Omni Microphone`.

Select this check box to baffle the back of the antenna element. In this case, the antenna responses to all azimuth angles beyond ±90° from broadside are set to zero. Define the broadside direction as 0° azimuth angle and 0° elevation angle.

Ports

Note The block input and output ports correspond to the input and output parameters described in the `step` method of the underlying System object. See link at the bottom of this page.

Port	Description	Supported Data Types
`X`	Input signal. The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.	Double-precision floating point
`Steer`	Initial estimate of broadside DOA angles.	Double-precision floating point
`Ang`	Estimated broadside DOA angles.	Double-precision floating point

Version History

Introduced in R2014b

See Also

`phased.SumDifferenceMonopulseTracker`

URA Sum and Difference Monopulse

Sum-and-difference monopulse for URA

Library

Direction of Arrival (DOA)

`phaseddoalib`

Description

The URA Sum-and-Difference Monopulse block estimates the direction of arrival of a narrowband signal on a uniform rectangular array (URA) based on an initial guess using a sum-and-difference monopulse algorithm. The block obtains the difference steering vector by phase-reversing the latter half of the sum steering vector.

Parameters

Signal Propagation speed (m/s)

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can use the function `physconst` to specify the speed of light.

Operating frequency (Hz)

Specify the operating frequency of the system, in hertz, as a positive scalar.

Number of bits in phase shifters

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Simulate using

Block simulation method, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster than they would in interpreted execution. You can run repeated executions without recompiling. However, if you change any block parameters, then the block automatically recompiles before execution.

When setting this parameter, you must take into account the overall model simulation mode. The table shows how the **Simulate using** parameter interacts with the overall simulation mode.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see "Choosing a Simulation Mode" (Simulink).

Array Parameters

Specify sensor array as

Specify a ULA sensor array directly or by using a MATLAB expression.

Types

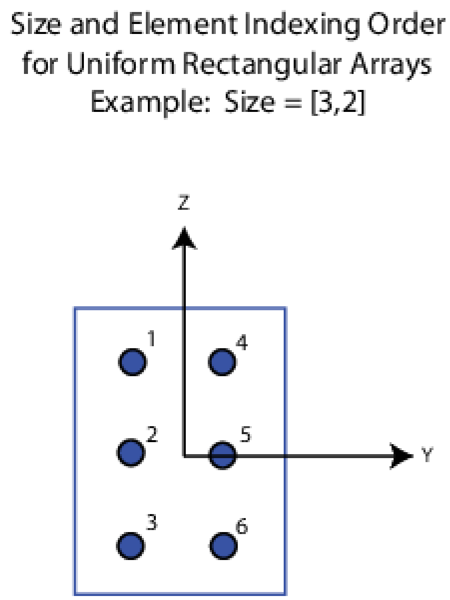
`Array (no subarrays)`
`MATLAB expression`

Array size

Specify the size of the array as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.

Elements are indexed from top to bottom along a column and continuing to the next columns from left to right. In this figure, an **Array size** of `[3,2]` produces an array has three rows and two columns.



Element spacing

Specify the element spacing of the array, in meters, as a 1-by-2 vector or a scalar. If **Element spacing** is a 1-by-2 vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumns]`. For a discussion of these quantities, see `phased.URA`. If **Element spacing** is a scalar, the spacings between rows and columns are equal.

Element lattice

Specify the element lattice as one of `Rectangular` or `Triangular`.

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular` — Shifts the even-row elements of a rectangular lattice toward the positive-row axis direction. The elements are shifted a distance of half the element spacing along the row.

Array normal

This parameter appears when you set **Geometry** to `URA` or `UCA`. Specify the **Array normal** as `x`, `y`, or `z`. All URA and UCA array elements are placed in the *yz*, *zx*, or *xy*-planes, respectively, of the array coordinate system.

Taper

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

Specify element tapering as a complex-valued scalar or complex-valued M-by-N matrix. In this matrix, M is the number of elements along the z-axis, and N is the number of elements along the y-axis. M and N correspond to the values of `[NumberofRows, NumberOfColumns]` in the **Array size** matrix. If `Taper` is a scalar, the same weight is applied to each element. If the value of **Taper** is a matrix, a weight from the matrix is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Expression

A valid MATLAB expression containing a constructor for a uniform rectangular array, for example, `phased.URA`.

Sensor Array Tab: Element Parameters

Element type

Specify antenna or microphone type as

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Exponent of cosine pattern

This parameter appears when you set **Element type** to `Cosine Antenna`.

Specify the exponent of the cosine pattern as a scalar or a 1-by-2 vector. You must specify all values as non-negative real numbers. When you set **Exponent of cosine pattern** to a scalar, both the azimuth direction cosine pattern and the elevation direction cosine pattern are raised to the specified value. When you set **Exponent of cosine pattern** to a 1-by-2 vector, the first element is the exponent for the azimuth direction cosine pattern and the second element is the exponent for the elevation direction cosine pattern.

Operating frequency range (Hz)

This parameter appears when **Element type** is set to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Specify the operating frequency range, in hertz, of the antenna element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The antenna element has no response outside the specified frequency range.

Operating frequency vector (Hz)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify the frequencies, in Hz, at which to set the antenna and microphone frequency responses as a 1-by-*L* row vector of increasing values. Use **Frequency responses** to set the frequency responses. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of **Operating frequency vector (Hz)**.

Frequency responses (dB)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify this parameter as the frequency response of an antenna or microphone, in decibels, for the frequencies defined by **Operating frequency vector (Hz)**. Specify **Frequency responses (dB)** as a 1-by-*L* vector matching the dimensions of the vector specified in **Operating frequency vector (Hz)**.

Input Pattern Coordinate System

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Azimuth angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Azimuth angles must lie between –180° and 180° and be in strictly increasing order.

Elevation angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90° and be in strictly increasing order.

Phi Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Phi angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Theta Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Theta angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 180° and be in strictly increasing order.

Magnitude pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Phase pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

MatchArrayNormal

This parameter appears when the **Element type** is set to `Custom Antenna`.

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Polar pattern frequencies (Hz)

This parameter appears when the **Element type** is set to `Custom Microphone`.

Specify the measuring frequencies of the polar patterns as a 1-by-*M* vector. The measuring frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** parameter. Frequency units are in Hz.

Polar pattern angles (deg)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the measuring angles of the polar patterns, as a 1-by-*N* vector. The angles are measured from the central pickup axis of the microphone, and must be between –180° and 180°, inclusive.

Polar pattern (dB)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the magnitude of the microphone element polar pattern as an *M*-by-*N* matrix. *M* is the number of measuring frequencies specified in **Polar pattern frequencies (Hz)**. *N* is the number of measuring angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. Assume that the pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. Assume that the polar pattern is symmetric around the central axis. You can construct the microphone's response pattern in 3-D space from the polar pattern.

Baffle the back of the element

This check box appears only when the **Element type** parameter is set to `Isotropic Antenna` or `Omni Microphone`.

Select this check box to baffle the back of the antenna element. In this case, the antenna responses to all azimuth angles beyond ±90° from broadside are set to zero. Define the broadside direction as 0° azimuth angle and 0° elevation angle.

Ports

Note The block input and output ports correspond to the input and output parameters described in the `step` method of the underlying System object. See link at the bottom of this page.

Port	Description	Supported Data Types
`X`	Input signal. The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.	Double-precision floating point
`Steer`	Initial estimate of arrival directions.	Double-precision floating point
`Ang`	Estimate of arrival directions.	Double-precision floating point

Version History

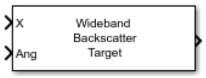
Introduced in R2014b

See Also

`phased.SumDifferenceMonopulseTracker2D`

Wideband Backscatter Radar Target

Backscatter wideband signals from radar target



Libraries:
Phased Array System Toolbox / Environment and Target

Description

The Wideband Backscatter Radar Target block models the monostatic reflection of nonpolarized wideband electromagnetic signals from a radar target. The target radar cross-section (RCS) model includes four Swerling target fluctuation models and a nonfluctuating model. You can model several targets simultaneously by specifying multiple radar cross-section RCS matrices.

Ports

Input

X — Wideband incident nonpolarized signal
N-by-*1* complex-valued vector | *N*-by-*M* complex-valued matrix

Wideband incident nonpolarized signal, specified as an *N*-by-*1* complex-valued vector or an *N*-by-*M* complex-valued matrix. The quantity *N* is the number of signal samples, and *M* is the number of independent signals reflecting off the target. Each column contains an independent signal to be reflected from the target.

The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.

Data Types: `double`

Ang — Incident signal direction
2-by-1 real-valued column vector | 2-by-*M* real-valued column matrix

Incident signal direction, specified as a 2-by-1 real-valued column vector or a 2-by-*M* real-valued column matrix. *M* is the number of signals reflecting from the target. Each column of `Ang` specifies the incident direction of the corresponding signal in the form of an `[AzimuthAngle;ElevationAngle]` pair. Units are degrees. The number of columns in `Ang` must match the number of independent signals in `X`.

Example: `[30;45]`

Data Types: `double`

Update — Switch to update RCS
`false` | `true`

Switch to update RCS fluctuation model values, specified as `false` or `true`. When `Update` is `true`, the RCS value is updated. If `Update` is `false`, the RCS remains unchanged.

Dependencies

To enable this port, set the **Fluctuation model** drop-down menu to `Swerling1`, `Swerling2`, `Swerling3`, or `Swerling4`.

Output

Port_1 — Narrowband reflected signal
1-by-*M* complex-valued vector | *N*-by-*M* complex-valued matrix

Wideband nonpolarized signal, specified as an 1-by-*M* complex-valued vector or a *N*-by-*M* complex-valued matrix. Each column contains an independent signal reflected from the target.

The quantity *N* is the number of signal samples and *M* is the number of signals reflecting off the target. Each column corresponds to a different reflecting angle.

The output port contains signal samples arriving at the signal destination within the current input time frame. When the propagation time from source to destination exceeds the current time frame duration, the output does not contain all contributions from the input of the current time frame.

Parameters

Backscatter pattern frequency vector (Hz) — Wideband backscatter pattern frequencies

`[0,1e20]` (default) | real-valued row vector of positive values in strictly increasing order

Specify the frequencies used in the RCS matrix. The elements of this vector must be in strictly increasing order. The target has no response outside this frequency range. Frequencies are defined with respect to the physical frequency band, not the baseband. Frequency units are in Hz.

Data Types: `double`

Azimuth angles (deg) — Azimuth angles

`[-180:180]` (default) | 1-by-*P* real-valued row vector | *P*-by-1 real-valued column vector

Azimuth angles used to define the angular coordinates of each column of the matrices specified by the **RCS pattern (m^2)** parameter. Specify the azimuth angles as a length *P* vector. *P* must be greater than two. Angle units are in degrees.

Example: `[-45:0.1:45]`

Data Types: `double`

Elevation angles (deg) — Elevation angles
`[-90:90]` (default) | 1-by-*Q* real-valued row vector | *Q*-by-1 real-valued column vector

Elevation angles used to define the angular coordinates of each row of the matrices specified by the **RCS pattern (m^2)** parameter. Specify the elevation angles as a length *Q* vector. *Q* must be greater than two. Angle units are in degrees.

Example: `[-30:0.1:30]`

Data Types: `double`

RCS pattern (m^2) — Radar cross-section pattern
`ones(181,361)` (default) | *Q*-by-*P* real-valued matrix | *Q*-by-*P*-by-*K* real-valued array | 1-by-*P*-by-*K* real-valued array | *K*-by-*P* real-valued matrix

Radar cross-section pattern, specified as a real-valued matrix or array.

<table>
<tr><th>Dimensions</th><th>Application</th></tr>
<tr><td>Q-by-P matrix</td><td>Specifies a matrix of RCS values as a function of Q elevation angles and P azimuth angles. The same RCS matrix is used for all frequencies.</td></tr>
<tr><td>Q-by-P-by-K array</td><td>Specifies an array of RCS patterns as a function of Q elevation angles, P azimuth angles, and K frequencies. If K = 1, the RCS pattern is equivalent to a Q-by-P matrix.</td></tr>
<tr><td>1-by-P-by-K array</td><td rowspan="2">Specifies a matrix of RCS values as a function of P azimuth angles and K frequencies. These dimension formats apply when there is only one elevation angle.</td></tr>
<tr><td>K-by-P matrix</td></tr>
</table>

- *Q* is the length of the vector specified by the **Elevation angles (deg)** parameter.
- *P* is the length of the vector specified by the **Azimuth angles (deg)** parameter.
- *K* is the number of frequencies specified by the **Backscatter pattern frequency vector (Hz)** parameter.

You can specify patterns for *L* targets by putting *L* patterns into a cell array. All patterns must have the same dimensions. The value of *L* must match the column dimensions of the signals passed as input into the block. You can, however, use one pattern to model *L* multiple targets.

RCS units are in square meters.

Example: `[1,2;2,1]`

Data Types: `double`

Fluctuation model — Target fluctuation model
`Nonfluctuating` (default) | `Swerling1` | `Swerling2` | `Swerling3` | `Swerling4`

Target fluctuation model, specified as `Nonfluctuating`, `Swerling1`, `Swerling2`, `Swerling3`, or `Swerling4`. If you set this parameter to a value other than `Nonfluctuating`, you must pass either `true` or `false` into the **Update** `Update` port.

Propagation speed (m/s) — Signal propagation speed
`physconst('LightSpeed')` (default) | positive scalar

Signal propagation speed, specified as a real-valued positive scalar. The default value of the speed of light is the value returned by `physconst('LightSpeed')`.

Data Types: `double`

Operating frequency (Hz) — Signal carrier frequency
`300.0e6` (default) | positive real-valued scalar

Signal carrier frequency, specified as a positive real-valued scalar. Units are in hertz.

Inherit sample rate — Inherit sample rate from upstream blocks
on (default) | off

Select this parameter to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Data Types: `Boolean`

Sample rate (Hz) — Sampling rate of signal
`1e6` (default) | positive real-valued scalar

Specify the signal sampling rate as a positive scalar. Units are in Hz.

Dependencies

To enable this parameter, clear the **Inherit sample rate** check box.

Data Types: `double`

Number of subbands — Number of processing subbands
`64` (default) | positive integer

Number of processing subbands, specified as a positive integer.

Example: `128`

Simulate using — Block simulation method
`Interpreted Execution` (default) | `Code Generation`

Block simulation, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster with generated code than in interpreted execution. You can run repeated executions without recompiling, but if you change any block parameters, then the block automatically recompiles before execution.

This table shows how the **Simulate using** parameter affects the overall simulation behavior.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Programmatic Use
Block Parameter:`SimulateUsing`
Type:enum
Values:`Interpreted Execution`, `Code Generation`
Default:`Interpreted Execution`

Version History
Introduced in R2016b

See Also

Objects
`phased.BackscatterRadarTarget` | `phased.RadarTarget`

Blocks
Backscatter Radar Target | Radar Target

Wideband Free Space

Wideband free space environment

Library

Environment and Target

`phasedenvlib`

Description

The Wideband Free Space Channel block propagates the signal from one point to another in space. The block models propagation time, free space propagation loss and Doppler shift. The block assumes that the propagation speed is much greater than the target or array speed in which case the stop-and-hop model is valid.

When propagating a signal in free-space to an object and back, you have the choice of either using a single block to compute a two-way free space propagation delay or two blocks to perform one-way propagation delays in each direction. Because the free-space propagation delay is not necessarily an integer multiple of the sampling interval, it may turn out that the total round trip delay in samples when you use a two-way propagation block differs from the delay in samples when you use two one-way propagation blocks. For this reason, it is recommended that, when possible, you use a single two-way propagation block.

Parameters

Signal Propagation speed (m/s)

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can use the function `physconst` to specify the speed of light.

Signal carrier frequency (Hz)

Specify the carrier frequency of the signal in hertz of the narrowband signal as a positive scalar.

Number of subbands

The number of subbands used for subband processing, specified as a positive integer.

Perform two-way propagation

Select this check box to perform round-trip propagation between the origin and destination. Otherwise the block performs one-way propagation from the origin to the destination.

Inherit sample rate

Select this check box to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Sample rate (Hz)

Specify the signal sampling rate (in hertz) as a positive scalar. This parameter appears only when the **Inherit sample rate** parameter is not selected.

Maximum one-way propagation distance (m)

The maximum distance , in meters, between the origin and the destination as a positive scalar. Amplitudes of any signals that propagate beyond this distance will be set to zero.

Simulate using

Block simulation method, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster than they would in interpreted execution. You can run repeated executions without recompiling. However, if you change any block parameters, then the block automatically recompiles before execution.

When setting this parameter, you must take into account the overall model simulation mode. The table shows how the **Simulate using** parameter interacts with the overall simulation mode.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Ports

Note The block input and output ports correspond to the input and output parameters described in the `step` method of the underlying System object. See link at the bottom of this page.

Port	Description	Supported Data Types
`X`	Input signal.	Double-precision floating point
`Pos1`	Signal source position.	Double-precision floating point
`Pos2`	Signal destination position.	Double-precision floating point

Port	Description	Supported Data Types
`Vel1`	Signal source velocity.	Double-precision floating point
`Vel2`	Signal destination velocity.	Double-precision floating point
`Out`	Propagated signal.	Double-precision floating point

Algorithms

When the origin and destination are stationary relative to each other, the block output can be written as $y(t) = x(t - \tau)/L$. The quantity τ is the delay and L is the propagation loss. The delay is computed from $\tau = R/c$ where R is the propagation distance and c is the propagation speed. The free space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far-field of the transmitting element or array. In the near-field, the free-space path loss formula is not valid and can result in losses smaller than one, equivalent to a signal gain. For this reason, the loss is set to unity for range values, $R \le \lambda/4\pi$.

When there is relative motion between the origin and destination, the processing also introduces a frequency shift. This shift corresponds to the Doppler shift between the origin and destination. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The parameter v is the relative speed of the destination with respect to the origin.

Version History

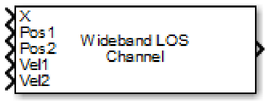
Introduced in R2015b

See Also

`phased.WidebandFreeSpace`

Wideband LOS Channel

Wideband line-of-sight propagation channel



Library

Environment and Target

`phasedenvlib`

Description

The Wideband LOS Channel block propagates signals from one point in space to multiple points or from multiple points back to one point via line-of-sight (LOS) channels. The block models propagation time, free-space propagation loss, Doppler shift, and atmospheric as well as weather loss. The block assumes that the propagation speed is much greater than the object's speed in which case the stop-and-hop model is valid.

When propagating a signal in an LOS channel to an object and back, you have the choice of either using a single block to compute two-way LOS channel propagation delay or two blocks to perform one-way propagation delays in each direction. Because the LOS channel propagation delay is not necessarily an integer multiple of the sampling interval, it may turn out that the total round trip delay in samples when you use a two-way propagation block differs from the delay in samples when you use two one-way propagation blocks. For this reason, it is recommended that, when possible, you use a single two-way propagation block.

Parameters

Signal Propagation speed (m/s)

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can use the function `physconst` to specify the speed of light.

Signal carrier frequency (Hz)

Specify the carrier frequency of the signal in hertz of the narrowband signal as a positive scalar.

Number of subbands

The number of subbands used for subband processing, specified as a positive integer.

Specify atmospheric parameters

Select this check box to enable atmospheric attenuation modeling.

Temperature (degrees Celsius)

Ambient atmospheric temperature, specified as a real-valued scalar. Units are degrees Celsius. This parameter appears when you select the **Specify atmospheric parameters** check box. Units are degrees Celsius.

Dry air pressure (Pa)

Atmospheric dry air pressure, specified as a positive real-valued scalar. Units are Pascals (Pa). The value 101325 for this property corresponds to one standard atmosphere. This parameter appears when you select the **Specify atmospheric parameters** check box.

Water vapour density (g/m^3)

Atmospheric water vapor density, specified as a positive real-valued scalar. Units are gm/m^3. This parameter appears when you select the **Specify atmospheric parameters** check box.

Liquid water density (g/m^3)

Liquid water density of fog or clouds, specified as a non-negative real-valued scalar. Units are gm/m^3. Typical values for liquid water density are 0.05 for medium fog and 0.5 for thick fog. This parameter appears when you select the **Specify atmospheric parameters** check box.

Rain rate (mm/hr)

Rainfall rate, specified as a non-negative real-valued scalar. Units are in mm/hour. This parameter appears when you select the **Specify atmospheric parameters** check box.

Perform two-way propagation

Select this check box to perform round-trip propagation between the origin and destination. Otherwise the block performs one-way propagation from the origin to the destination.

Inherit sample rate

Select this check box to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Sample rate (Hz)

Specify the signal sampling rate (in hertz) as a positive scalar. This parameter appears only when the **Inherit sample rate** parameter is not selected.

Maximum one-way propagation distance (m)

The maximum distance between the signal origin and the destination, specified as a positive scalar. Units are in meters. Amplitudes of any signals that propagate beyond this distance will be set to zero.

Simulate using

Block simulation method, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster than they would in interpreted execution. You can run repeated executions without recompiling. However, if you change any block parameters, then the block automatically recompiles before execution.

When setting this parameter, you must take into account the overall model simulation mode. The table shows how the **Simulate using** parameter interacts with the overall simulation mode.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	Normal	Accelerator	Rapid Accelerator
Interpreted Execution	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
Code Generation	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Ports

Note The block input and output ports correspond to the input and output parameters described in the `step` method of the underlying System object. See link at the bottom of this page.

Port	Description	Supported Data Types
X	Input signal.	Double-precision floating point
Pos1	Signal source position.	Double-precision floating point
Pos2	Signal destination position.	Double-precision floating point
Vel1	Signal source velocity.	Double-precision floating point
Vel2	Signal destination velocity.	Double-precision floating point
Out	Propagated signal.	Double-precision floating point

More About

Attenuation and Loss Factors

Attenuation or path loss in the Wideband LOS channel consists of four components. $L = L_{fsp}L_gL_cL_r$, where

- L_{fsp} is the free-space path attenuation
- L_g is the atmospheric path attenuation
- L_c is the fog and cloud path attenuation
- L_r is the rain path attenuation

Each component is in magnitude units, not in dB.

Propagation Delay, Doppler, and Free-Space Path Loss

When the origin and destination are stationary relative to each other, you can write the output signal of a free-space channel as $Y(t) = x(t-\tau)/L_{fsp}$. The quantity τ is the signal delay and L_{fsp} is the free-space path loss. The delay τ is given by R/c, where R is the propagation distance and c is the propagation speed. The free-space path loss is given by

$$L_{fsp} = \frac{(4\pi R)^2}{\lambda^2},$$

where λ is the signal wavelength.

This formula assumes that the target is in the far field of the transmitting element or array. In the near field, the free-space path loss formula is not valid and can result in a loss smaller than one, equivalent to a signal gain. Therefore, the loss is set to unity for range values, $R \le \lambda/4\pi$.

When the origin and destination have relative motion, the processing also introduces a Doppler frequency shift. The frequency shift is v/λ for one-way propagation and $2v/\lambda$ for two-way propagation. The quantity v is the relative speed of the destination with respect to the origin.

Atmospheric Gas Attenuation Model

This model calculates the attenuation of signals that propagate through atmospheric gases.

Electromagnetic signals attenuate when they propagate through the atmosphere. This effect is due primarily to the absorption resonance lines of oxygen and water vapor, with smaller contributions coming from nitrogen gas. The model also includes a continuous absorption spectrum below 10 GHz. The ITU model *Recommendation ITU-R P.676-10: Attenuation by atmospheric gases* is used. The model computes the specific attenuation (attenuation per kilometer) as a function of temperature, pressure, water vapor density, and signal frequency. The atmospheric gas model is valid for frequencies from 1–1000 GHz and applies to polarized and nonpolarized fields.

The formula for specific attenuation at each frequency is

$$\gamma = \gamma_o(f) + \gamma_w(f) = 0.1820 f N''(f)\,.$$

The quantity *N"()* is the imaginary part of the complex atmospheric refractivity and consists of a spectral line component and a continuous component:

$$N''(f) = \sum_i S_i F_i + N''_D(f)$$

The spectral component consists of a sum of discrete spectrum terms composed of a localized frequency bandwidth function, $F(f)_i$, multiplied by a spectral line strength, S_i. For atmospheric oxygen, each spectral line strength is

$$S_i = a_1 \times 10^{-7}\left(\frac{300}{T}\right)^3 \exp\left[a_2(1 - \left(\frac{300}{T}\right)\right]P\,.$$

For atmospheric water vapor, each spectral line strength is

$$S_i = b_1 \times 10^{-1}\left(\frac{300}{T}\right)^{3.5} \exp\left[b_2(1 - \left(\frac{300}{T}\right)\right]W\,.$$

P is the dry air pressure, W is the water vapor partial pressure, and T is the ambient temperature. Pressure units are in hectoPascals (hPa) and temperature is in degrees Kelvin. The water vapor partial pressure, W, is related to the water vapor density, ρ, by

$$W = \frac{\rho T}{216.7}\,.$$

The total atmospheric pressure is $P + W$.

For each oxygen line, S_i depends on two parameters, a_1 and a_2. Similarly, each water vapor line depends on two parameters, b_1 and b_2. The ITU documentation cited at the end of this section contains tabulations of these parameters as functions of frequency.

The localized frequency bandwidth functions $F_i(f)$ are complicated functions of frequency described in the ITU references cited below. The functions depend on empirical model parameters that are also tabulated in the reference.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length, R. Then, the total attenuation is $L_g = R(\gamma_o + \gamma_w)$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Fog and Cloud Attenuation Model

This model calculates the attenuation of signals that propagate through fog or clouds.

Fog and cloud attenuation are the same atmospheric phenomenon. The ITU model, *Recommendation ITU-R P.840-6: Attenuation due to clouds and fog* is used. The model computes the specific attenuation (attenuation per kilometer), of a signal as a function of liquid water density, signal frequency, and temperature. The model applies to polarized and nonpolarized fields. The formula for specific attenuation at each frequency is

$$\gamma_c = K_l(f)M,$$

where M is the liquid water density in gm/m^3. The quantity $K_l(f)$ is the specific attenuation coefficient and depends on frequency. The cloud and fog attenuation model is valid for frequencies 10–1000 GHz. Units for the specific attenuation coefficient are (dB/km)/(g/m^3).

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the path length R. Total attenuation is $L_c = R\gamma_c$.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands, and apply narrowband attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Rainfall Attenuation Model

This model calculates the attenuation of signals that propagate through regions of rainfall. Rain attenuation is a dominant fading mechanism and can vary from location-to-location and from year-to-year.

Electromagnetic signals are attenuated when propagating through a region of rainfall. Rainfall attenuation is computed according to the ITU rainfall model *Recommendation ITU-R P.838-3: Specific attenuation model for rain for use in prediction methods*. The model computes the specific attenuation (attenuation per kilometer) of a signal as a function of rainfall rate, signal frequency, polarization, and path elevation angle. The specific attenuation, γ_R, is modeled as a power law with respect to rain rate

$$\gamma_R = kR^\alpha,$$

where R is rain rate. Units are in mm/hr. The parameter k and exponent α depend on the frequency, the polarization state, and the elevation angle of the signal path. The specific attenuation model is valid for frequencies from 1–1000 GHz.

To compute the total attenuation for narrowband signals along a path, the function multiplies the specific attenuation by the an effective propagation distance, d_{eff}. Then, the total attenuation is $L = d_{\mathrm{eff}}\gamma_{\mathrm{R}}$.

The effective distance is the geometric distance, d, multiplied by a scale factor

$$r = \frac{1}{0.477d^{0.633}R_{0.01}^{0.073\alpha}f^{0.123} - 10.579(1 - \exp(-0.024d))}$$

where f is the frequency. The article *Recommendation ITU-R P.530-17 (12/2017): Propagation data and prediction methods required for the design of terrestrial line-of-sight systems* presents a complete discussion for computing attenuation.

The rain rate, R, used in these computations is the long-term statistical rain rate, $R_{0.01}$. This is the rain rate that is exceeded 0.01% of the time. The calculation of the statistical rain rate is discussed in *Recommendation ITU-R P.837-7 (06/2017): Characteristics of precipitation for propagation modelling*. This article also explains how to compute the attenuation for other percentages from the 0.01% value.

You can apply the attenuation model to wideband signals. First, divide the wideband signal into frequency subbands and apply attenuation to each subband. Then, sum all attenuated subband signals into the total attenuated signal.

Subband Frequency Processing

Subband processing decomposes a wideband signal into multiple subbands and applies narrowband processing to the signal in each subband. The signals for all subbands are summed to form the output signal.

When using wideband frequency System objects or blocks, you specify the number of subbands, N_{B}, in which to decompose the wideband signal. Subband center frequencies and widths are automatically computed from the total bandwidth and number of subbands. The total frequency band is centered on the carrier or operating frequency, f_c. The overall bandwidth is given by the sample rate, f_s. Frequency subband widths are $\Delta f = f_s/N_{\mathrm{B}}$. The center frequencies of the subbands are

$$f_m = \begin{cases} f_c - \dfrac{f_s}{2} + (m-1)\Delta f, & N_B \text{ even} \\ f_c - \dfrac{(N_B - 1)f_s}{2N_B} + (m-1)\Delta f, & N_B \text{ odd} \end{cases}, \quad m = 1, \ldots, N_B$$

Some System objects let you obtain the subband center frequencies as output when you run the object. The returned subband frequencies are ordered consistently with the ordering of the discrete Fourier transform. Frequencies above the carrier appear first, followed by frequencies below the carrier.

Version History

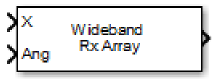
Introduced in R2016a

See Also

Objects
`phased.WidebandLOSChannel` | `phased.LOSChannel`

Wideband Receive Array

Wideband receive array



Library

Transmitters and Receivers

`phasedtxrxlib`

Description

The Wideband Receive Array block receives wideband plane waves incident on the elements of a sensor array. The block divides the input signal into subbands and then applies a phase shift in each subband according to the incident direction. The resulting subband signals are then combined to form the output.

Parameters

Signal Propagation speed (m/s)

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can use the function `physconst` to specify the speed of light.

Inherit sample rate

Select this check box to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Sample rate (Hz)

Specify the signal sampling rate (in hertz) as a positive scalar. This parameter appears only when the **Inherit sample rate** parameter is not selected.

Assume modulated input

Select this check this box to indicate that the input signal is demodulated at a carrier frequency.

Carrier frequency

This parameter appears when the **Assume modulated input** check box is selected. The parameter specifies the carrier frequency, in hertz, as a positive scalar.

Number of subbands

Number of processing subbands, specified as a positive integer.

Sensor gain measure

Sensor gain measure, specified as `dB` or `dBi`.

- When you set this parameter to `dB`, the input signal power is scaled by the sensor power pattern (in dB) at the corresponding direction and then combined.

- When you set this parameter to `dBi`, the input signal power is scaled by the directivity pattern (in dBi) at the corresponding direction and then combined. This option is useful when you want to compare results with the values computed by the radar equation that uses dBi to specify the antenna gain. The computation using the `dBi` option is expensive as it requires an integration over all directions to compute the total radiated power of the sensor. The default value is `dB`.

Enable weights input

Select this check box to specify array weights using the input port `W`. The input port appears only when this box is checked.

Simulate using

Block simulation method, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster than they would in interpreted execution. You can run repeated executions without recompiling. However, if you change any block parameters, then the block automatically recompiles before execution.

When setting this parameter, you must take into account the overall model simulation mode. The table shows how the **Simulate using** parameter interacts with the overall simulation mode.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

<table>
<tr><th rowspan="2">Block Simulation</th><th colspan="3">Simulation Behavior</th></tr>
<tr><th><code>Normal</code></th><th><code>Accelerator</code></th><th><code>Rapid Accelerator</code></th></tr>
<tr><td><code>Interpreted Execution</code></td><td>The block executes using the MATLAB interpreter.</td><td>The block executes using the MATLAB interpreter.</td><td rowspan="2">Creates a standalone executable from the model.</td></tr>
<tr><td><code>Code Generation</code></td><td>The block is compiled.</td><td>All blocks in the model are compiled.</td></tr>
</table>

For more information, see “Choosing a Simulation Mode” (Simulink).

Array Parameters

Specify sensor array as

Specify sensor element or sensor array. A sensor array can also contain subarrays or be a partitioned array. This parameter can also be expressed as a MATLAB expression.

Types

`Single element`
`Array (no subarrays)`
`Partitioned array`
`Replicated subarray`
`MATLAB expression`

Geometry

Specify the array geometry as one of the following:

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements

Number of array elements.

Number of array elements, specified as a positive integer. This parameter appears when the **Geometry** is set to `ULA` or `UCA`. If **Sensor Array** has a `Replicated subarray` option, this parameter applies to the subarray.

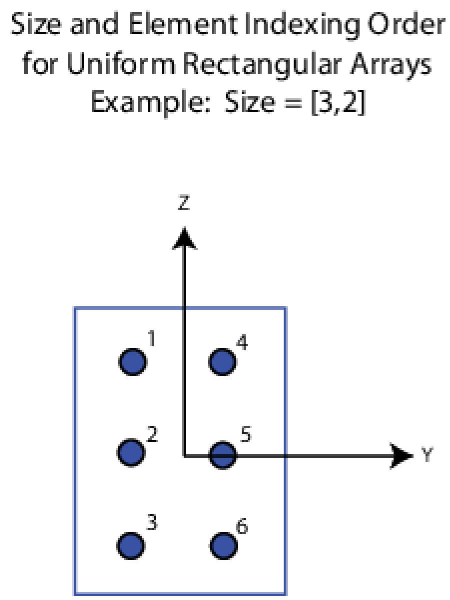
Array size

This parameter appears when **Geometry** is set to `URA`. When **Sensor Array** is set to `Replicated subarray`, this parameter applies to the subarrays.

Specify the size of the array as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.

For a URA, elements are indexed from top to bottom along a column and continuing to the next columns from left to right. In this figure, an **Array size** of `[3,2]` produces an array of three rows and two columns.



Element spacing (m)

This parameter appears when **Geometry** is set to ULA or URA. When **Sensor Array** has the Replicated subarray option, this parameter applies to the subarrays.

- For a ULA, specify the spacing, in meters, between two adjacent elements in the array as a scalar.
- For a URA, specify the element spacing of the array, in meters, as a 1-by-2 vector or a scalar. If **Element spacing** is a 1-by-2 vector, the vector has the form [SpacingBetweenRows,SpacingBetweenColumns]. For a discussion of these quantities, see phased.URA. If **Element spacing** is a scalar, the spacings between rows and columns are equal.

Array axis

This parameter appears when the **Geometry** parameter is set to ULA or when the block only supports a ULA array geometry. Specify the array axis as x, y, or z. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Array normal

This parameter appears when you set **Geometry** to URA or UCA. Specify the **Array normal** as x, y, or z. All URA and UCA array elements are placed in the *yz*, *zx*, or *xy*-planes, respectively, of the array coordinate system.

Radius of UCA (m)

Radius of a uniform circular array specified as a positive scalar. Units are meters.

This parameter appears when the **Geometry** is set to UCA.

Taper

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

This parameter applies to all array types, but when you set **Sensor Array** to Replicated subarray, this parameter applies to subarrays.

- For a `ULA` or `UCA`, specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.
- For a `URA`, specify element tapering as a complex-valued scalar or complex-valued *M*-by-*N* matrix. In this matrix, *M* is the number of elements along the *z*-axis, and *N* is the number of elements along the *y*-axis. *M* and *N* correspond to the values of `[NumberofArrayRows,NumberOfArrayColumns]` in the **Array size** matrix. If `Taper` is a scalar, the same weight is applied to each element. If **Taper** is a matrix, a weight from the matrix is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.
- For a `Conformal Array`, specify element tapering as a complex-valued scalar or complex-valued 1-by-*N* vector. In this vector, *N* is the number of elements in the array as determined by the size of the **Element positions** vector. If **Taper** is a scalar, the same weight is applied to each element. If the value of **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Element lattice

This parameter appears when **Geometry** is set to `URA`. When **Sensor Array** is set to `Replicated subarray`, this parameter applies to the subarray.

Specify the element lattice as `Rectangular` or `Triangular`

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular`— Shifts the even-row elements of a rectangular lattice toward the positive-row axis direction. The displacement is one-half the element spacing along the row dimension.

Element positions (m)

This parameter appears when **Geometry** is set to `Conformal Array`. When **Sensor Array** is set to `Replicated subarray`, this parameter applies to subarrays.

Specify the positions of conformal array elements as a 3-by-*N* matrix, where *N* is the number of elements in the conformal array. Each column of **Element positions (m)** represents the position of a single element, in the form `[x;y;z]`, in the array's local coordinate system. The local coordinate system has its origin at an arbitrary point. Units are in meters.

Element normals (deg)

This parameter appears when **Geometry** is set to `Conformal Array`. When **Sensor Array** is set to `Replicated subarray`, this parameter applies to subarrays.

Specify the normal directions of the elements in a conformal array as a 2-by-*N* matrix or a 2-by-1 column vector in degrees. The variable *N* indicates the number of elements in the array. If **Element normals (deg)** is a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]`, with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If **Element normals (deg)** is a 2-by-1 column vector, the vector specifies the same pointing direction for all elements in the array.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. You can combine translation, azimuth rotation, and elevation rotation transformations. However, you cannot use transformations that require rotation about the normal.

Subarray definition matrix

This parameter appears when **Specify sensor array as** is set to `Partitioned array`.

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix corresponds to a subarray and each entry in the row indicates whether or not an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray is its geometric center. **Subarray definition matrix** and **Geometry** determine the geometric center.

Subarray steering method

This parameter appears when the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`.

Specify the subarray steering method as either

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Phase shifter frequency (Hz)

This parameter appears when you set **Sensor array** to `Partitioned array` or `Replicated subarray` and you set **Subarray steering method** to `Phase`.

Specify the operating frequency, in hertz, of phase shifters to perform subarray steering as a positive scalar.

Number of bits in phase shifters

This parameter appears when you set **Sensor array** to `Partitioned array` or `Replicated subarray` and you set **Subarray steering method** to `Phase`.

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Subarrays layout

This parameter appears when you set **Sensor array** to `Replicated subarray`.

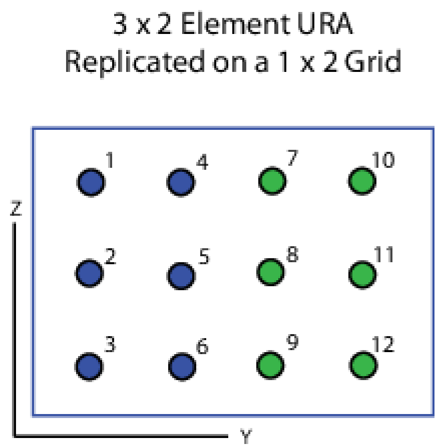
Specify the layout of the replicated subarrays as `Rectangular` or `Custom`.

Grid size

This parameter appears when you set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Rectangular subarray grid size, specified as a single positive integer or a positive integer-valued 1-by-2 row vector.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Grid spacing

This parameter appears when you set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Specify the rectangular grid spacing of subarrays as a real-valued positive scalar, a 1-by-2 row vector, or `Auto`. Grid spacing units are expressed in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Subarray positions (m)

This parameter appears when you set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Specify the positions of the subarrays in the custom grid as a 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray, in meters, in the array's local coordinate system. The coordinates are expressed in the form `[x; y; z]`.

Subarray normals

This parameter appears when you set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth; elevation]`. Each angle is in degrees and is defined in the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Expression

A valid MATLAB expression containing an array constructor, for example, `phased.URA`.

Sensor Array Tab: Element Parameters

Element type

Specify antenna or microphone type as

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Exponent of cosine pattern

This parameter appears when you set **Element type** to `Cosine Antenna`.

Specify the exponent of the cosine pattern as a scalar or a 1-by-2 vector. You must specify all values as non-negative real numbers. When you set **Exponent of cosine pattern** to a scalar, both the azimuth direction cosine pattern and the elevation direction cosine pattern are raised to the specified value. When you set **Exponent of cosine pattern** to a 1-by-2 vector, the first element is the exponent for the azimuth direction cosine pattern and the second element is the exponent for the elevation direction cosine pattern.

Operating frequency range (Hz)

This parameter appears when **Element type** is set to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Specify the operating frequency range, in hertz, of the antenna element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The antenna element has no response outside the specified frequency range.

Operating frequency vector (Hz)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify the frequencies, in Hz, at which to set the antenna and microphone frequency responses as a 1-by-*L* row vector of increasing values. Use **Frequency responses** to set the frequency responses. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of **Operating frequency vector (Hz)**.

Frequency responses (dB)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify this parameter as the frequency response of an antenna or microphone, in decibels, for the frequencies defined by **Operating frequency vector (Hz)**. Specify **Frequency responses (dB)** as a 1-by-*L* vector matching the dimensions of the vector specified in **Operating frequency vector (Hz)**.

Input Pattern Coordinate System

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Azimuth angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Azimuth angles must lie between –180° and 180° and be in strictly increasing order.

Elevation angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90° and be in strictly increasing order.

Phi Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Phi angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Theta Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Theta angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 180° and be in strictly increasing order.

Magnitude pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Phase pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

MatchArrayNormal

This parameter appears when the **Element type** is set to `Custom Antenna`.

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the URA and UCA arrays.

Polar pattern frequencies (Hz)

This parameter appears when the **Element type** is set to `Custom Microphone`.

Specify the measuring frequencies of the polar patterns as a 1-by-*M* vector. The measuring frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** parameter. Frequency units are in Hz.

Polar pattern angles (deg)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the measuring angles of the polar patterns, as a 1-by-*N* vector. The angles are measured from the central pickup axis of the microphone, and must be between –180° and 180°, inclusive.

Polar pattern (dB)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the magnitude of the microphone element polar pattern as an *M*-by-*N* matrix. *M* is the number of measuring frequencies specified in **Polar pattern frequencies (Hz)**. *N* is the number of measuring angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. Assume that the pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. Assume that the polar pattern is symmetric around the central axis. You can construct the microphone's response pattern in 3-D space from the polar pattern.

Baffle the back of the element

This check box appears only when the **Element type** parameter is set to `Isotropic Antenna` or `Omni Microphone`.

Select this check box to baffle the back of the antenna element. In this case, the antenna responses to all azimuth angles beyond ±90° from broadside are set to zero. Define the broadside direction as 0° azimuth angle and 0° elevation angle.

Ports

Note The block input and output ports correspond to the input and output parameters described in the `step` method of the underlying System object. See link at the bottom of this page.

Port	Description	Supported Data Types
`X`	Arriving signals input port The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.	Double-precision floating point
`Ang`	Incident directions of signals input port.	Double-precision floating point

Port	Description	Supported Data Types
W	Array or subarray weights input port. To enable this port, select the **Enable weights input** check box.	Double-precision floating point
WS	Subarray element weights input port. To enable this port select Custom from the **Subarray steering method** pull down menu.	
Steer	Steering angle input port. To enable this port, select Phase or Time from the **Subarray steering method** pull down menu.	Double-precision floating point
Out	Collected signals	Double-precision floating point

Version History

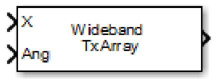
Introduced in R2014b

See Also

phased.WidebandCollector

Wideband Transmit Array

Wideband transmit array



Library

Transmitters and Receivers

`phasedtxrxlib`

Description

The Wideband Transmit Array block transmits wideband plane waves from the elements of a sensor array. The block divides the transmitted signals into subbands and then applies a phase shift for each subband according to the radiating direction. The resulting subband signals are then combined to form the output.

Parameters

Signal Propagation speed (m/s)

Specify the propagation speed of the signal, in meters per second, as a positive scalar. You can use the function `physconst` to specify the speed of light.

Inherit sample rate

Select this check box to inherit the sample rate from upstream blocks. Otherwise, specify the sample rate using the **Sample rate (Hz)** parameter.

Sample rate (Hz)

Specify the signal sampling rate (in hertz) as a positive scalar. This parameter appears only when the **Inherit sample rate** parameter is not selected.

Assume modulated input

Select this check this box to indicate that the input signal is demodulated at a carrier frequency.

Carrier frequency

This parameter appears when the **Assume modulated input** check box is selected. The parameter specifies the carrier frequency, in hertz, as a positive scalar.

Number of subbands

The number of subbands used for subband processing, specified as a positive integer.

Sensor gain measure

Sensor gain measure, specified as `dB` or `dBi`.

- When you set this parameter to `dB`, the input signal power is scaled by the sensor power pattern (in dB) at the corresponding direction and then combined.

- When you set this parameter to `dBi`, the input signal power is scaled by the directivity pattern (in dBi) at the corresponding direction and then combined. This option is useful when you want to compare results with the values computed by the radar equation that uses dBi to specify the antenna gain. The computation using the `dBi` option is expensive as it requires an integration over all directions to compute the total radiated power of the sensor. The default value is `dB`.

Enable weights input

Select this check box to specify array weights using the input port `W`. The input port appears only when this box is checked.

Simulate using

Block simulation method, specified as `Interpreted Execution` or `Code Generation`. If you want your block to use the MATLAB interpreter, choose `Interpreted Execution`. If you want your block to run as compiled code, choose `Code Generation`. Compiled code requires time to compile but usually runs faster.

Interpreted execution is useful when you are developing and tuning a model. The block runs the underlying System object in MATLAB. You can change and execute your model quickly. When you are satisfied with your results, you can then run the block using `Code Generation`. Long simulations run faster than they would in interpreted execution. You can run repeated executions without recompiling. However, if you change any block parameters, then the block automatically recompiles before execution.

When setting this parameter, you must take into account the overall model simulation mode. The table shows how the **Simulate using** parameter interacts with the overall simulation mode.

When the Simulink model is in `Accelerator` mode, the block mode specified using **Simulate using** overrides the simulation mode.

Acceleration Modes

Block Simulation	Simulation Behavior		
	`Normal`	`Accelerator`	`Rapid Accelerator`
`Interpreted Execution`	The block executes using the MATLAB interpreter.	The block executes using the MATLAB interpreter.	Creates a standalone executable from the model.
`Code Generation`	The block is compiled.	All blocks in the model are compiled.	

For more information, see “Choosing a Simulation Mode” (Simulink).

Array Parameters

Specify sensor array as

Specify sensor element or sensor array. A sensor array can also contain subarrays or be a partitioned array. This parameter can also be expressed as a MATLAB expression.

Types

`Single element`
`Array (no subarrays)`
`Partitioned array`
`Replicated subarray`
`MATLAB expression`

Geometry

Specify the array geometry as one of the following:

- `ULA` — Uniform linear array
- `URA` — Uniform rectangular array
- `UCA` — Uniform circular array
- `Conformal Array` — arbitrary element positions

Number of elements

Number of array elements.

Number of array elements, specified as a positive integer. This parameter appears when the **Geometry** is set to `ULA` or `UCA`. If **Sensor Array** has a `Replicated subarray` option, this parameter applies to the subarray.

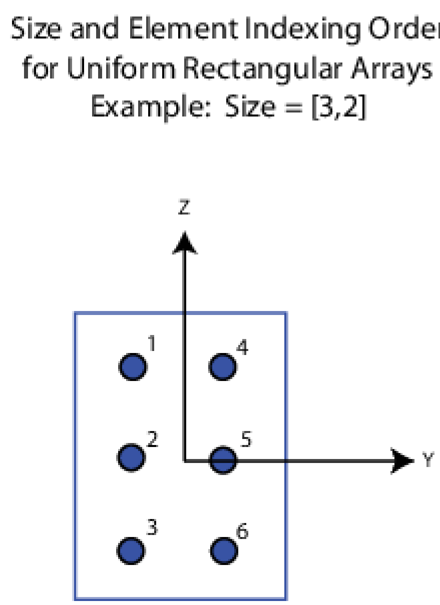
Array size

This parameter appears when **Geometry** is set to `URA`. When **Sensor Array** is set to `Replicated subarray`, this parameter applies to the subarrays.

Specify the size of the array as a positive integer or 1-by-2 vector of positive integers.

- If **Array size** is a 1-by-2 vector, the vector has the form `[NumberOfArrayRows,NumberOfArrayColumns]`.
- If **Array size** is an integer, the array has the same number of rows and columns.

For a URA, elements are indexed from top to bottom along a column and continuing to the next columns from left to right. In this figure, an **Array size** of `[3,2]` produces an array of three rows and two columns.



Element spacing (m)

This parameter appears when **Geometry** is set to `ULA` or `URA`. When **Sensor Array** has the `Replicated subarray` option, this parameter applies to the subarrays.

- For a `ULA`, specify the spacing, in meters, between two adjacent elements in the array as a scalar.
- For a `URA`, specify the element spacing of the array, in meters, as a 1-by-2 vector or a scalar. If **Element spacing** is a 1-by-2 vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumns]`. For a discussion of these quantities, see `phased.URA`. If **Element spacing** is a scalar, the spacings between rows and columns are equal.

Array axis

This parameter appears when the **Geometry** parameter is set to `ULA` or when the block only supports a ULA array geometry. Specify the array axis as `x`, `y`, or `z`. All ULA array elements are uniformly spaced along this axis in the local array coordinate system.

Array normal

This parameter appears when you set **Geometry** to `URA` or `UCA`. Specify the **Array normal** as `x`, `y`, or `z`. All URA and UCA array elements are placed in the *yz*, *zx*, or *xy*-planes, respectively, of the array coordinate system.

Radius of UCA (m)

Radius of a uniform circular array specified as a positive scalar. Units are meters.

This parameter appears when the **Geometry** is set to `UCA`.

Taper

Tapers, also known as element weights, are applied to sensor elements in the array. Tapers are used to modify both the amplitude and phase of the transmitted or received data.

This parameter applies to all array types, but when you set **Sensor Array** to `Replicated subarray`, this parameter applies to subarrays.

- For a ULA or UCA, specify element tapering as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements in the array. If **Taper** is a scalar, the same weight is applied to each element. If **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.
- For a URA, specify element tapering as a complex-valued scalar or complex-valued *M*-by-*N* matrix. In this matrix, *M* is the number of elements along the *z*-axis, and *N* is the number of elements along the *y*-axis. *M* and *N* correspond to the values of `[NumberofArrayRows,NumberOfArrayColumns]` in the **Array size** matrix. If `Taper` is a scalar, the same weight is applied to each element. If **Taper** is a matrix, a weight from the matrix is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.
- For a `Conformal Array`, specify element tapering as a complex-valued scalar or complex-valued 1-by-*N* vector. In this vector, *N* is the number of elements in the array as determined by the size of the **Element positions** vector. If **Taper** is a scalar, the same weight is applied to each element. If the value of **Taper** is a vector, a weight from the vector is applied to the corresponding sensor element. A weight must be applied to each element in the sensor array.

Element lattice

This parameter appears when **Geometry** is set to URA. When **Sensor Array** is set to `Replicated subarray`, this parameter applies to the subarray.

Specify the element lattice as `Rectangular` or `Triangular`

- `Rectangular` — Aligns all the elements in row and column directions.
- `Triangular`— Shifts the even-row elements of a rectangular lattice toward the positive-row axis direction. The displacement is one-half the element spacing along the row dimension.

Element positions (m)

This parameter appears when **Geometry** is set to `Conformal Array`. When **Sensor Array** is set to `Replicated subarray`, this parameter applies to subarrays.

Specify the positions of conformal array elements as a 3-by-*N* matrix, where *N* is the number of elements in the conformal array. Each column of **Element positions (m)** represents the position of a single element, in the form `[x;y;z]`, in the array's local coordinate system. The local coordinate system has its origin at an arbitrary point. Units are in meters.

Element normals (deg)

This parameter appears when **Geometry** is set to `Conformal Array`. When **Sensor Array** is set to `Replicated subarray`, this parameter applies to subarrays.

Specify the normal directions of the elements in a conformal array as a 2-by-*N* matrix or a 2-by-1 column vector in degrees. The variable *N* indicates the number of elements in the array. If **Element normals (deg)** is a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]`, with respect to the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal array. If **Element normals (deg)** is a 2-by-1 column vector, the vector specifies the same pointing direction for all elements in the array.

You can use the **Element positions (m)** and **Element normals (deg)** parameters to represent any arrangement in which pairs of elements differ by certain transformations. You can combine translation, azimuth rotation, and elevation rotation transformations. However, you cannot use transformations that require rotation about the normal.

Subarray definition matrix

This parameter appears when **Specify sensor array as** is set to `Partitioned array`.

Specify the subarray selection as an *M*-by-*N* matrix. *M* is the number of subarrays and *N* is the total number of elements in the array. Each row of the matrix corresponds to a subarray and each entry in the row indicates whether or not an element belongs to the subarray. When the entry is zero, the element does not belong the subarray. A nonzero entry represents a complex-valued weight applied to the corresponding element. Each row must contain at least one nonzero entry.

The phase center of each subarray is its geometric center. **Subarray definition matrix** and **Geometry** determine the geometric center.

Subarray steering method

This parameter appears when the **Specify sensor array as** parameter is set to `Partitioned array` or `Replicated subarray`.

Specify the subarray steering method as either

- `None`
- `Phase`
- `Time`
- `Custom`

Selecting `Phase` or `Time` opens the `Steer` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Selecting `Custom` opens the `WS` input port on the Narrowband Receive Array, Narrowband Transmit Array, Wideband Receive Array, Wideband Transmit Array blocks, Constant Gamma Clutter, and GPU Constant Gamma Clutter blocks.

Phase shifter frequency (Hz)

This parameter appears when you set **Sensor array** to `Partitioned array` or `Replicated subarray` and you set **Subarray steering method** to `Phase`.

Specify the operating frequency, in hertz, of phase shifters to perform subarray steering as a positive scalar.

Number of bits in phase shifters

This parameter appears when you set **Sensor array** to `Partitioned array` or `Replicated subarray` and you set **Subarray steering method** to `Phase`.

The number of bits used to quantize the phase shift component of beamformer or steering vector weights. Specify the number of bits as a non-negative integer. A value of zero indicates that no quantization is performed.

Subarrays layout

This parameter appears when you set **Sensor array** to `Replicated subarray`.

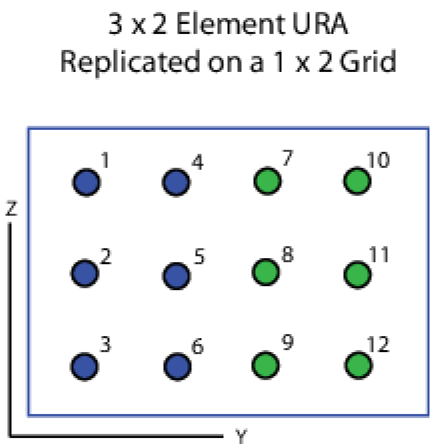
Specify the layout of the replicated subarrays as `Rectangular` or `Custom`.

Grid size

This parameter appears when you set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Rectangular subarray grid size, specified as a single positive integer or a positive integer-valued 1-by-2 row vector.

If **Grid size** is an integer scalar, the array has an equal number of subarrays in each row and column. If **Grid size** is a 1-by-2 vector of the form `[NumberOfRows, NumberOfColumns]`, the first entry is the number of subarrays along each column. The second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. The figure here shows how you can replicate a 3-by-2 URA subarray using a **Grid size** of `[1,2]`.



Grid spacing

This parameter appears when you set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Rectangular`.

Specify the rectangular grid spacing of subarrays as a real-valued positive scalar, a 1-by-2 row vector, or `Auto`. Grid spacing units are expressed in meters.

- If **Grid spacing** is a scalar, the spacing along the row and the spacing along the column is the same.
- If **Grid spacing** is a 1-by-2 row vector, the vector has the form `[SpacingBetweenRows,SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.
- If **Grid spacing** is set to `Auto`, replication preserves the element spacing of the subarray for both rows and columns while building the full array. This option is available only when you specify **Geometry** as `ULA` or `URA`.

Subarray positions (m)

This parameter appears when you set **Sensor array** to `Replicated subarray` and **Subarrays layout** to `Custom`.

Specify the positions of the subarrays in the custom grid as a 3-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix represents the position of a single subarray, in meters, in the array's local coordinate system. The coordinates are expressed in the form `[x; y; z]`.

Subarray normals

This parameter appears when you set the **Sensor array** parameter to `Replicated subarray` and the **Subarrays layout** to `Custom`.

Specify the normal directions of the subarrays in the array. This parameter value is a 2-by-*N* matrix, where *N* is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form `[azimuth; elevation]`. Each angle is in degrees and is defined in the local coordinate system.

You can use the **Subarray positions** and **Subarray normals** parameters to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Expression

A valid MATLAB expression containing an array constructor, for example, `phased.URA`.

Sensor Array Tab: Element Parameters

Element type

Specify antenna or microphone type as

- `Isotropic Antenna`
- `Cosine Antenna`
- `Custom Antenna`
- `Omni Microphone`
- `Custom Microphone`

Exponent of cosine pattern

This parameter appears when you set **Element type** to `Cosine Antenna`.

Specify the exponent of the cosine pattern as a scalar or a 1-by-2 vector. You must specify all values as non-negative real numbers. When you set **Exponent of cosine pattern** to a scalar, both the azimuth direction cosine pattern and the elevation direction cosine pattern are raised to the specified value. When you set **Exponent of cosine pattern** to a 1-by-2 vector, the first element is the exponent for the azimuth direction cosine pattern and the second element is the exponent for the elevation direction cosine pattern.

Operating frequency range (Hz)

This parameter appears when **Element type** is set to `Isotropic Antenna`, `Cosine Antenna`, or `Omni Microphone`.

Specify the operating frequency range, in hertz, of the antenna element as a 1-by-2 row vector in the form `[LowerBound,UpperBound]`. The antenna element has no response outside the specified frequency range.

Operating frequency vector (Hz)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify the frequencies, in Hz, at which to set the antenna and microphone frequency responses as a 1-by-*L* row vector of increasing values. Use **Frequency responses** to set the frequency responses. The antenna or microphone element has no response outside the frequency range specified by the minimum and maximum elements of **Operating frequency vector (Hz)**.

Frequency responses (dB)

This parameter appears when **Element type** is set to `Custom Antenna` or `Custom Microphone`.

Specify this parameter as the frequency response of an antenna or microphone, in decibels, for the frequencies defined by **Operating frequency vector (Hz)**. Specify **Frequency responses (dB)** as a 1-by-*L* vector matching the dimensions of the vector specified in **Operating frequency vector (Hz)**.

Input Pattern Coordinate System

Coordinate system of custom antenna pattern, specified `az-el` or `phi-theta`. When you specify `az-el`, use the **Azimuth angles (deg)** and **Elevations angles (deg)** parameters to specify the coordinates of the pattern points. When you specify `phi-theta`, use the **Phi angles (deg)** and **Theta angles (deg)** parameters to specify the coordinates of the pattern points.

Azimuth angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the azimuth angles at which to calculate the antenna radiation pattern as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Azimuth angles must lie between –180° and 180° and be in strictly increasing order.

Elevation angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `az-el`.

Specify the elevation angles at which to compute the radiation pattern as a 1-by-*Q* vector. *Q* must be greater than 2. Angle units are in degrees. Elevation angles must lie between –90° and 90° and be in strictly increasing order.

Phi Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Phi angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*P* row vector. *P* must be greater than 2. Angle units are in degrees. Phi angles must lie between 0° and 360° and be in strictly increasing order.

Theta Angles (deg)

This parameter appears when **Element type** is set to `Custom Antenna` and the **Input Pattern Coordinate System** parameter is set to `phi-theta`.

Theta angles of points at which to specify the antenna radiation pattern, specify as a 1-by-*Q* row vector. *Q* must be greater than 2. Angle units are in degrees. Theta angles must lie between 0° and 180° and be in strictly increasing order.

Magnitude pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Magnitude of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

Phase pattern (dB)

This parameter appears when the **Element type** is set to `Custom Antenna`.

Phase of the combined antenna radiation pattern, specified as a *Q*-by-*P* matrix or a *Q*-by-*P*-by-*L* array.

- When the **Input Pattern Coordinate System** parameter is set to `az-el`, *Q* equals the length of the vector specified by the **Elevation angles (deg)** parameter and *P* equals the length of the vector specified by the **Azimuth angles (deg)** parameter.
- When the **Input Pattern Coordinate System** parameter is set to `phi-theta`, *Q* equals the length of the vector specified by the **Theta Angles (deg)** parameter and *P* equals the length of the vector specified by the **Phi Angles (deg)** parameter.

The quantity *L* equals the length of the **Operating frequency vector (Hz)**.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

- If this parameter is a *Q*-by-*P* matrix, the same pattern is applied to *all* frequencies specified in the **Operating frequency vector (Hz)** parameter.
- If the value is a *Q*-by-*P*-by-*L* array, each *Q*-by-*P* page of the array specifies a pattern for the *corresponding* frequency specified in the **Operating frequency vector (Hz)** parameter.

MatchArrayNormal

This parameter appears when the **Element type** is set to `Custom Antenna`.

Select this check box to rotate the antenna element pattern to align with the array normal. When not selected, the element pattern is not rotated.

When the antenna is used in an antenna array and the **Input Pattern Coordinate System** parameter is `az-el`, selecting this check box rotates the pattern so that the *x*-axis of the element coordinate system points along the array normal. Not selecting uses the element pattern without the rotation.

When the antenna is used in an antenna array and **Input Pattern Coordinate System** is set to `phi-theta`, selecting this check box rotates the pattern so that the *z*-axis of the element coordinate system points along the array normal.

Use the parameter in conjunction with the **Array normal** parameter of the `URA` and `UCA` arrays.

Polar pattern frequencies (Hz)

This parameter appears when the **Element type** is set to `Custom Microphone`.

Specify the measuring frequencies of the polar patterns as a 1-by-*M* vector. The measuring frequencies lie within the frequency range specified by the **Operating frequency vector (Hz)** parameter. Frequency units are in Hz.

Polar pattern angles (deg)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the measuring angles of the polar patterns, as a 1-by-*N* vector. The angles are measured from the central pickup axis of the microphone, and must be between –180° and 180°, inclusive.

Polar pattern (dB)

This parameter appears when **Element type** is set to `Custom Microphone`.

Specify the magnitude of the microphone element polar pattern as an *M*-by-*N* matrix. *M* is the number of measuring frequencies specified in **Polar pattern frequencies (Hz)**. *N* is the number of measuring angles specified in **Polar pattern angles (deg)**. Each row of the matrix represents the magnitude of the polar pattern measured at the corresponding frequency specified in **Polar pattern frequencies (Hz)** and all angles specified in **Polar pattern angles (deg)**. Assume that the pattern is measured in the azimuth plane. In the azimuth plane, the elevation angle is 0° and the central pickup axis is 0° degrees azimuth and 0° degrees elevation. Assume that the polar pattern is symmetric around the central axis. You can construct the microphone's response pattern in 3-D space from the polar pattern.

Baffle the back of the element

This check box appears only when the **Element type** parameter is set to `Isotropic Antenna` or `Omni Microphone`.

Select this check box to baffle the back of the antenna element. In this case, the antenna responses to all azimuth angles beyond ±90° from broadside are set to zero. Define the broadside direction as 0° azimuth angle and 0° elevation angle.

Ports

Note The block input and output ports correspond to the input and output parameters described in the `step` method of the underlying System object. See link at the bottom of this page.

Port	Description	Supported Data Types
`X`	Radiated signals input port The size of the first dimension of the input matrix can vary to simulate a changing signal length. A size change can occur, for example, in the case of a pulse waveform with variable pulse repetition frequency.	Double-precision floating point
`Ang`	Radiating directions of signals input port.	Double-precision floating point

Port	Description	Supported Data Types
`W`	Array or subarray weights input port. To enable this port, select the **Enable weights input** check box.	Double-precision floating point
`WS`	Subarray element weights input port. To enable this port select `Custom` from the **Subarray steering method** pull down menu.	
`Steer`	Steering angle input port. To enable this port, select `Phase` or `Time` from the **Subarray steering method** pull down menu.	
`Out`	Radiated signals.	Double-precision floating point

Version History
Introduced in R2015b

See Also
`phased.WidebandRadiator`

4

Apps

Pulse Waveform Analyzer

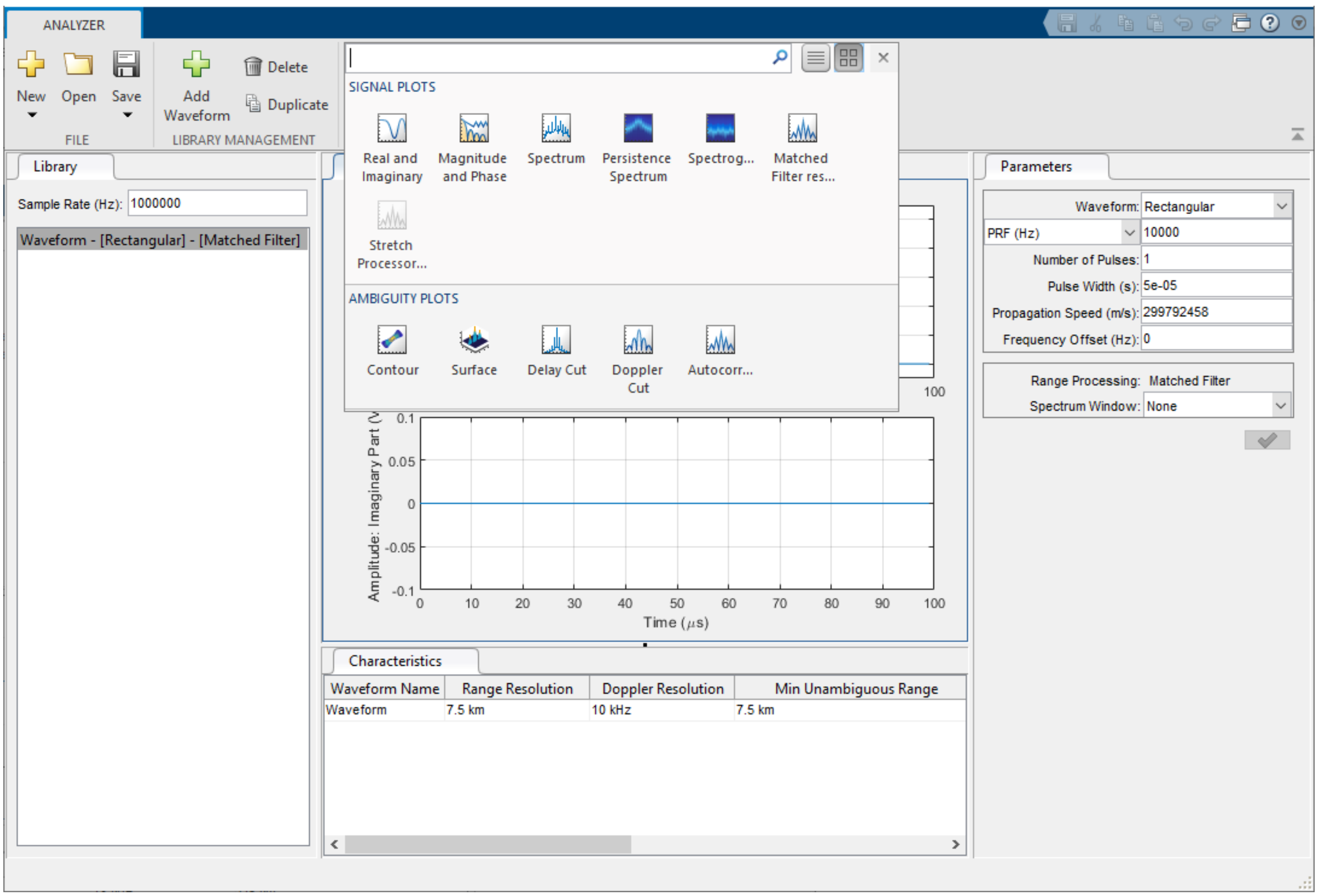
Analyze performance characteristics of pulsed, frequency-modulated, and phase-coded waveforms

Description

The **Pulse Waveform Analyzer** app lets you explore the properties of signals commonly used in radar. You can display 2-D and 3-D plots that let you visualize waveform time series and spectra.

The app lets you change waveform parameters and see how different parameter values affect the appearance and properties of the waveform. Waveform parameters include pulse repetition frequency (PRF), pulse duration, and bandwidth. The app displays basic waveform characteristics such as range resolution, Doppler resolution, and maximum range. When you launch the app, the **Real and Imaginary** and **Spectrum** tabs are shown by default. You can simultaneously overlay plots of multiple waveforms.

You can select different types of displays using this pull-down menu. You can also rearrange the tabs by using drag-and-drop to change the default layout.



The app lets you analyze these types of waveforms:

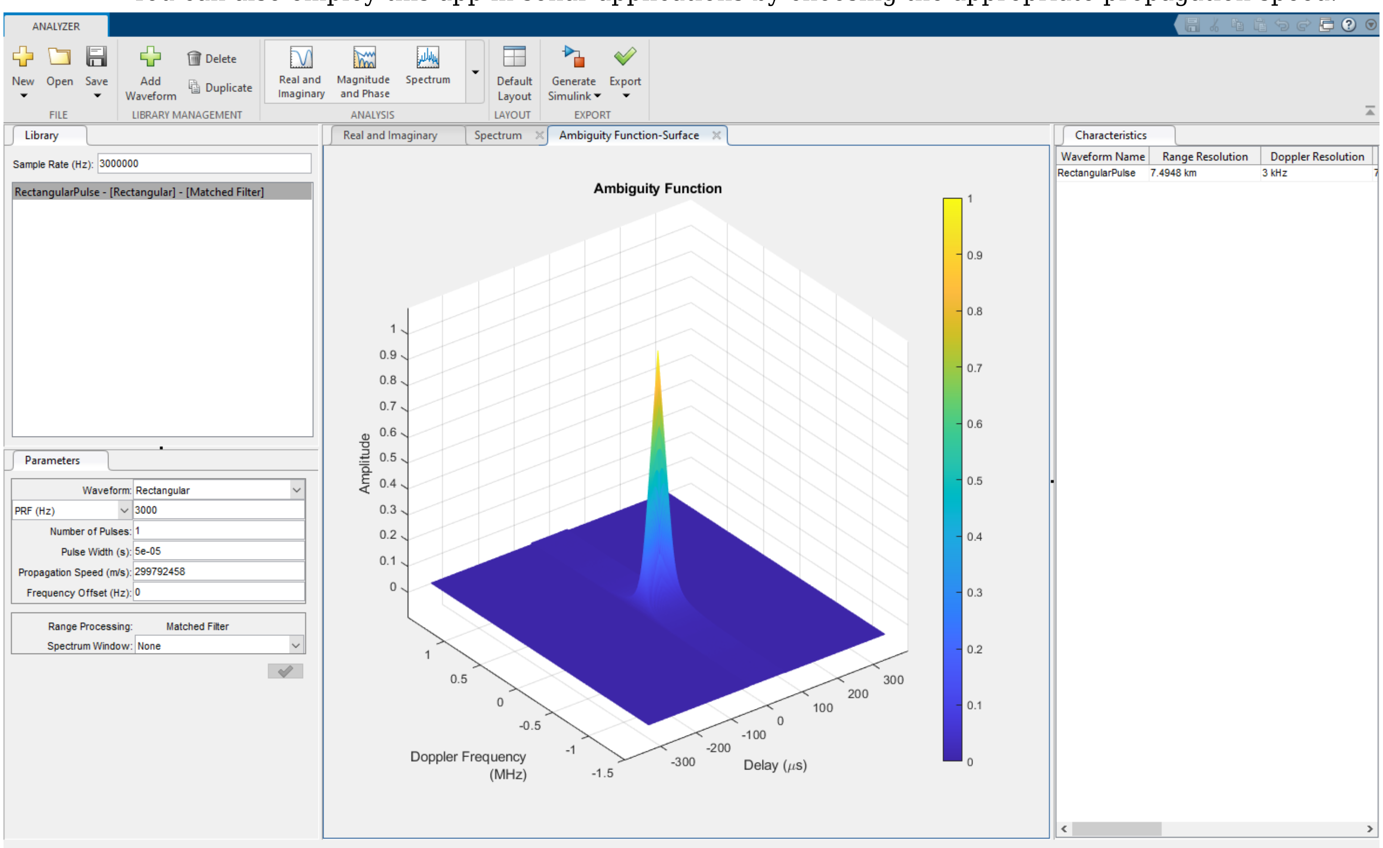
- Rectangular
- Linear frequency modulation (LFM)
- Stepped FM
- Phase-coded waveforms
- Frequency modulation constant waveform (FMCW)

You can export waveforms as workspace variables or files containing:

- Phased Array System Toolbox waveform objects such as `phased.LinearFMWaveform`.
- Radar Toolbox `pulseWaveformLibrary` objects.
- Radar Toolbox `pulseCompressionLibrary` objects.

You can use the waveform blocks, Matched Filter blocks, and Stretch Processor blocks in Simulink. You can also use the Pulse Waveform Library and Pulse Compression Library blocks available in Radar Toolbox.

You can also employ this app in sonar applications by choosing the appropriate propagation speed.



Open the Pulse Waveform Analyzer App

- MATLAB toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, select the app icon, or

- MATLAB command prompt: Enter `pulseWaveformAnalyzer`. For ways to use the app programmatically, see “Programmatic Use” on page 4-12.

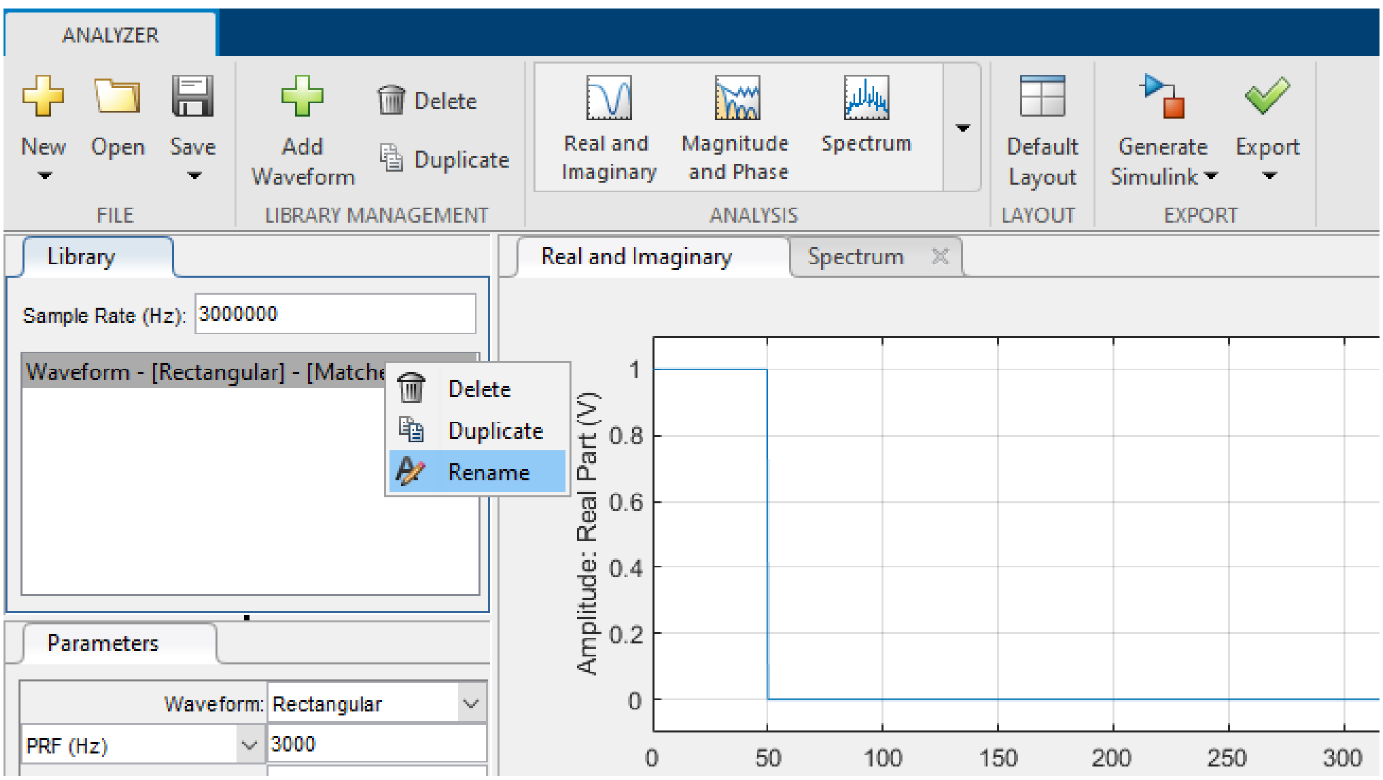
Examples

Rectangular Waveform

This example shows how to analyze a rectangular waveform. An ideal rectangular waveform jumps instantaneously to a constant value and stays there for some duration. Rearrange the **Parameters** and **Characteristics** tabs to make the plots larger.

When you open the app, the **Library** tab shows the default rectangular waveform and the center panel displays the waveform shape or spectrum. First, set the **Sample Rate (Hz)** to 3 MHz. The same sample rate applies to all waveforms that you analyze.

You can rename the waveform by right-clicking its name. Change the name to `RectangularPulse`.



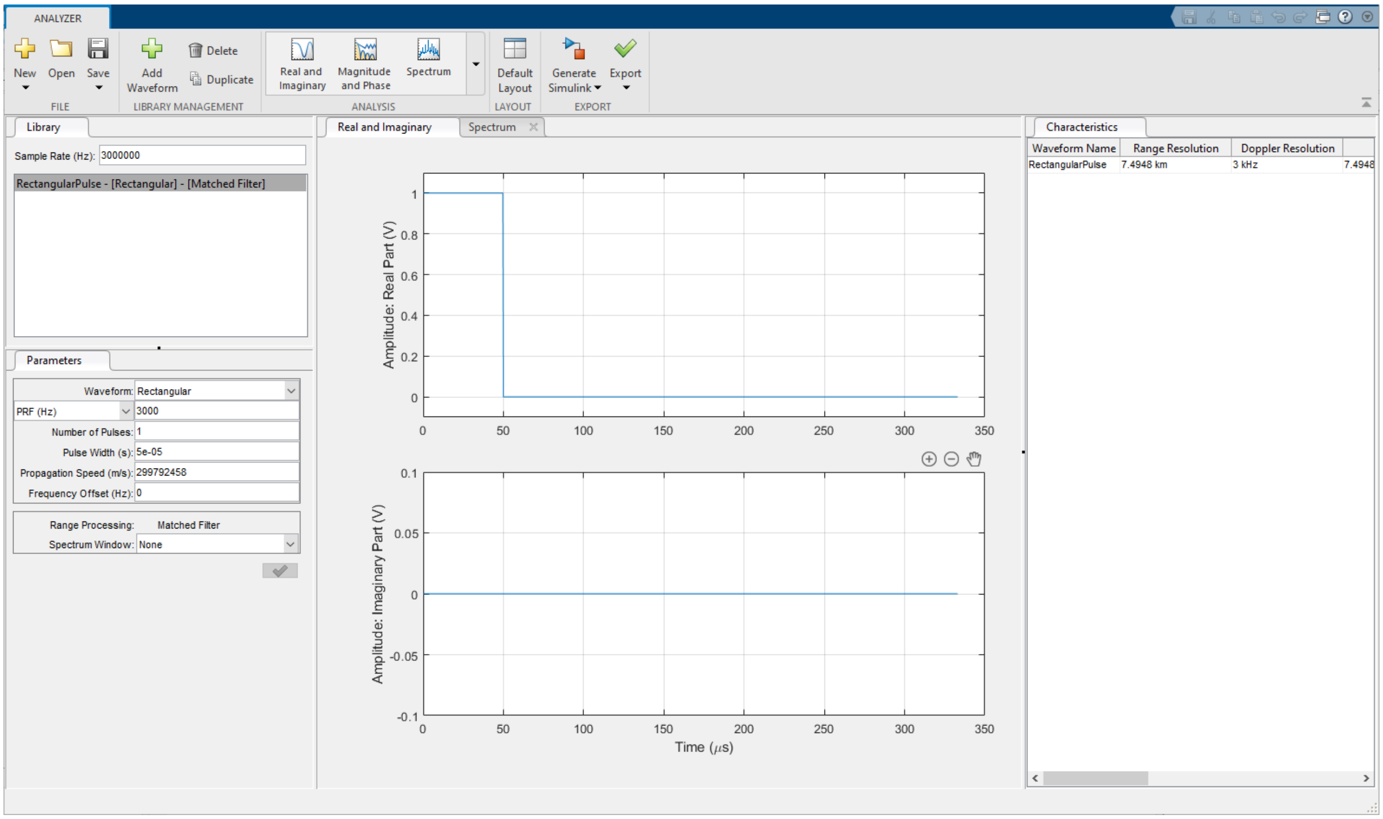
Design the pulse for a maximum range of 50 km. For this range, the time for a signal to propagate and return is 333 µs. Therefore, allow 333 µs between pulses, equivalent to a pulse repetition frequency (**PRF**) of 3000 Hz.

Set the **Pulse Width** to 50 µs.

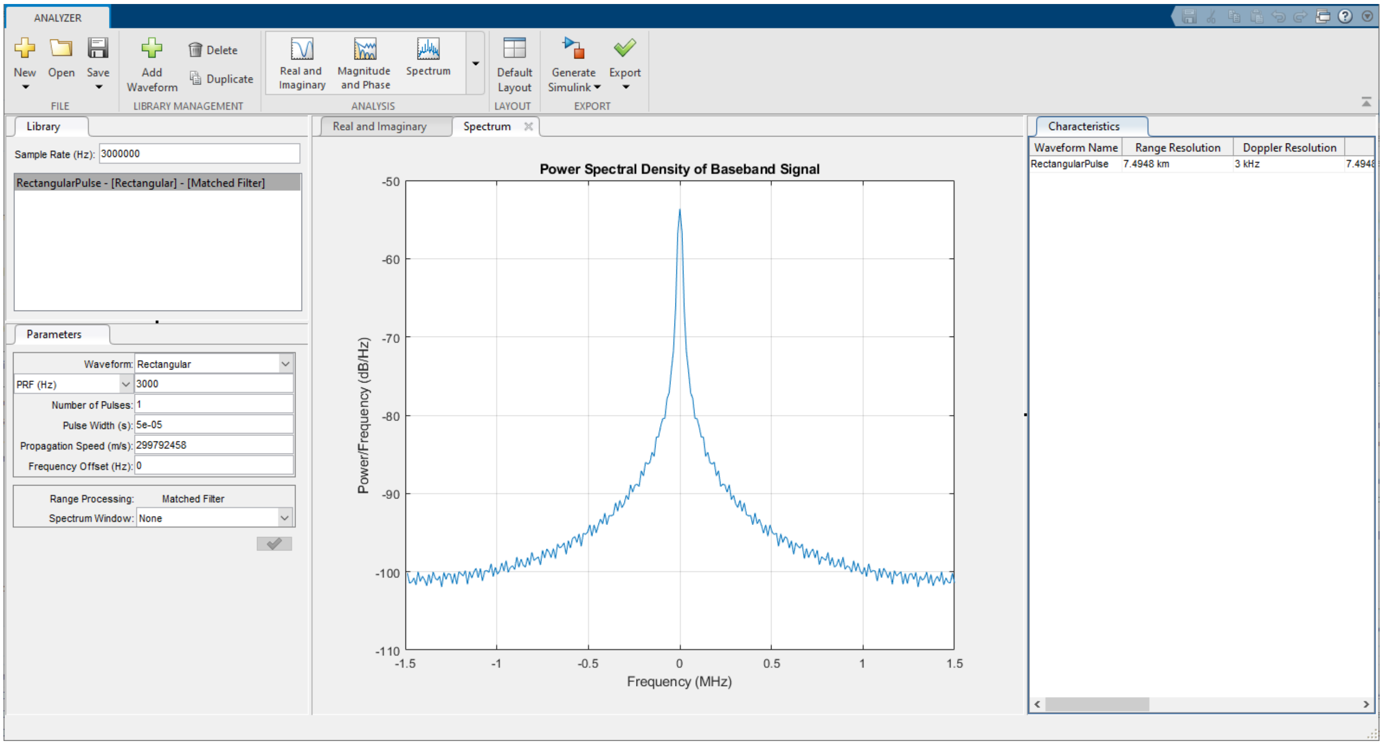
Change the value of the speed of light in the **Propagation Speed** field to a more precise value by entering `physconst('Lightspeed')`. You can use workspace variables and MATLAB functions in any editable field.

After you select the green check mark, the app displays a range resolution of approximately 7.5 km in the **Characteristics** tab. In this panel, you can scroll right to see other properties. The range resolution of a rectangular pulse is roughly 1/2 the pulse-width multiplied by the speed of light. The Doppler resolution is approximately the width of the Fourier transform of the pulse.

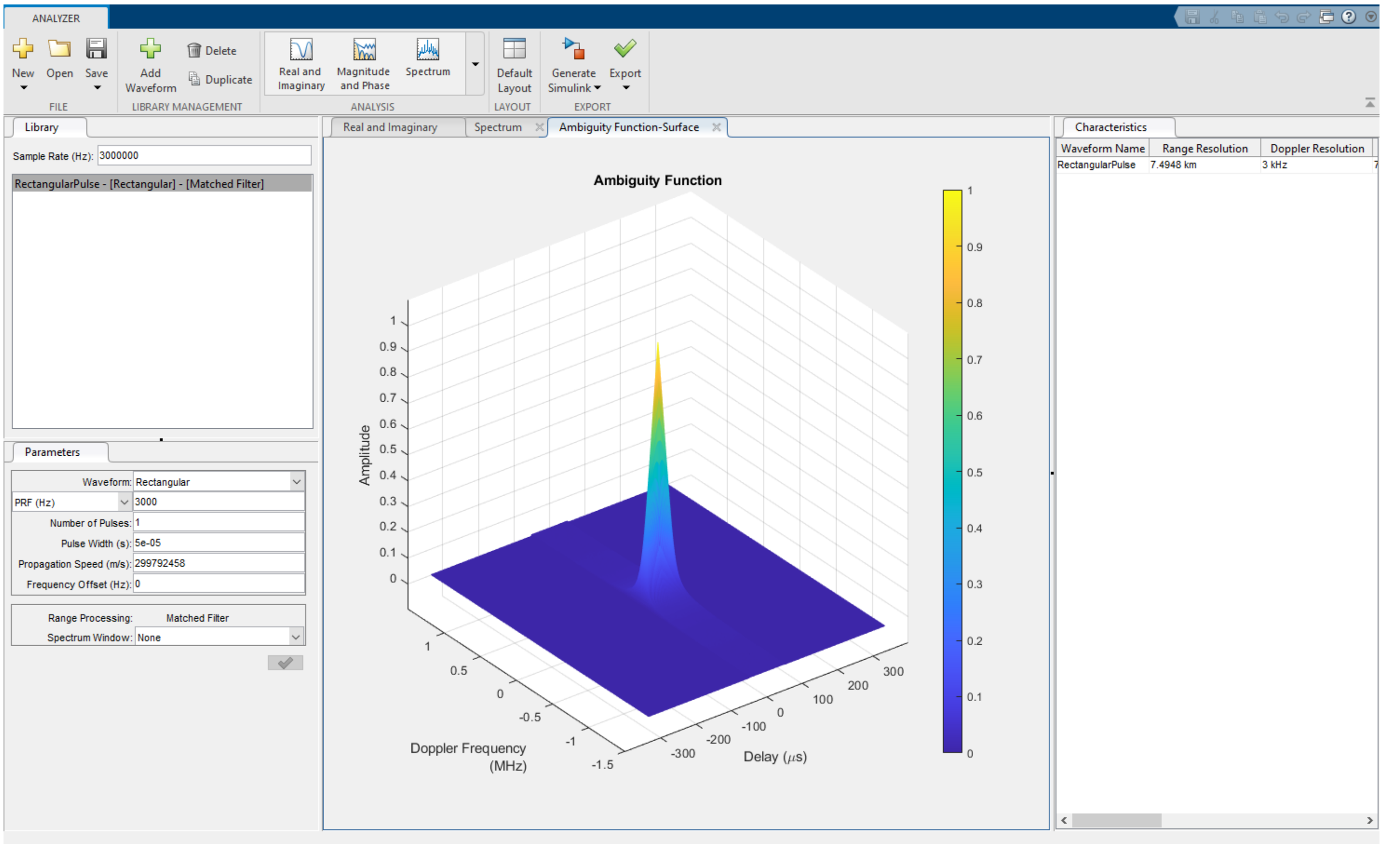
In the center panel of the window, select the **Real and Imaginary** tab to plot the waveform.



Select the **Spectrum** tab in the center panel of the window to show the power spectral density.



You can display the joint range-Doppler resolution by selecting **Surface** from the **Ambiguity Plots** menu.



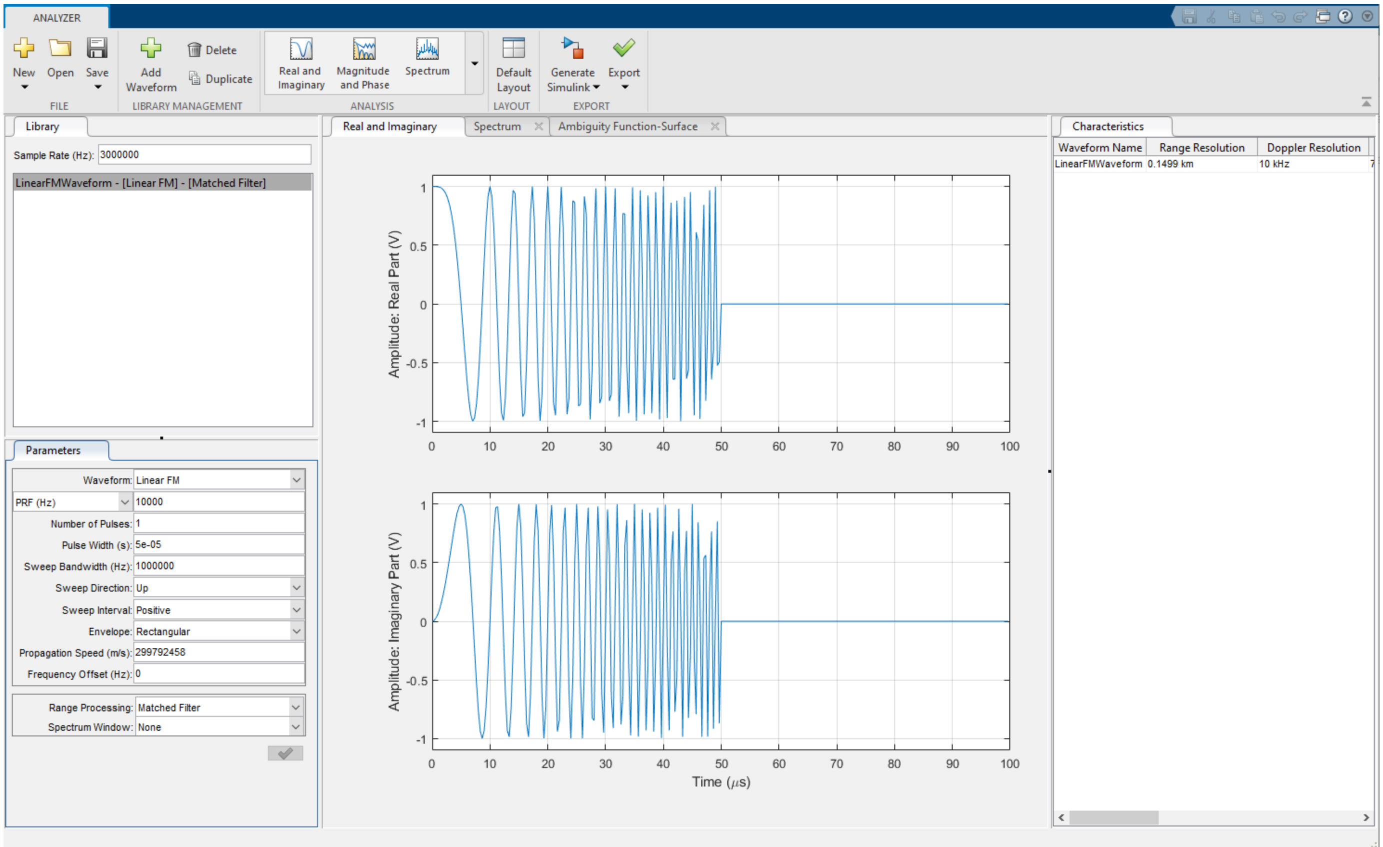
Linear FM Waveform

This example shows how to improve range resolution using a linear FM waveform. In the previous example, the range resolution of the rectangular pulse was poor, approximately 7.5 km. You can improve the range resolution by choosing a signal with a larger bandwidth. A good choice is a linear FM pulse.

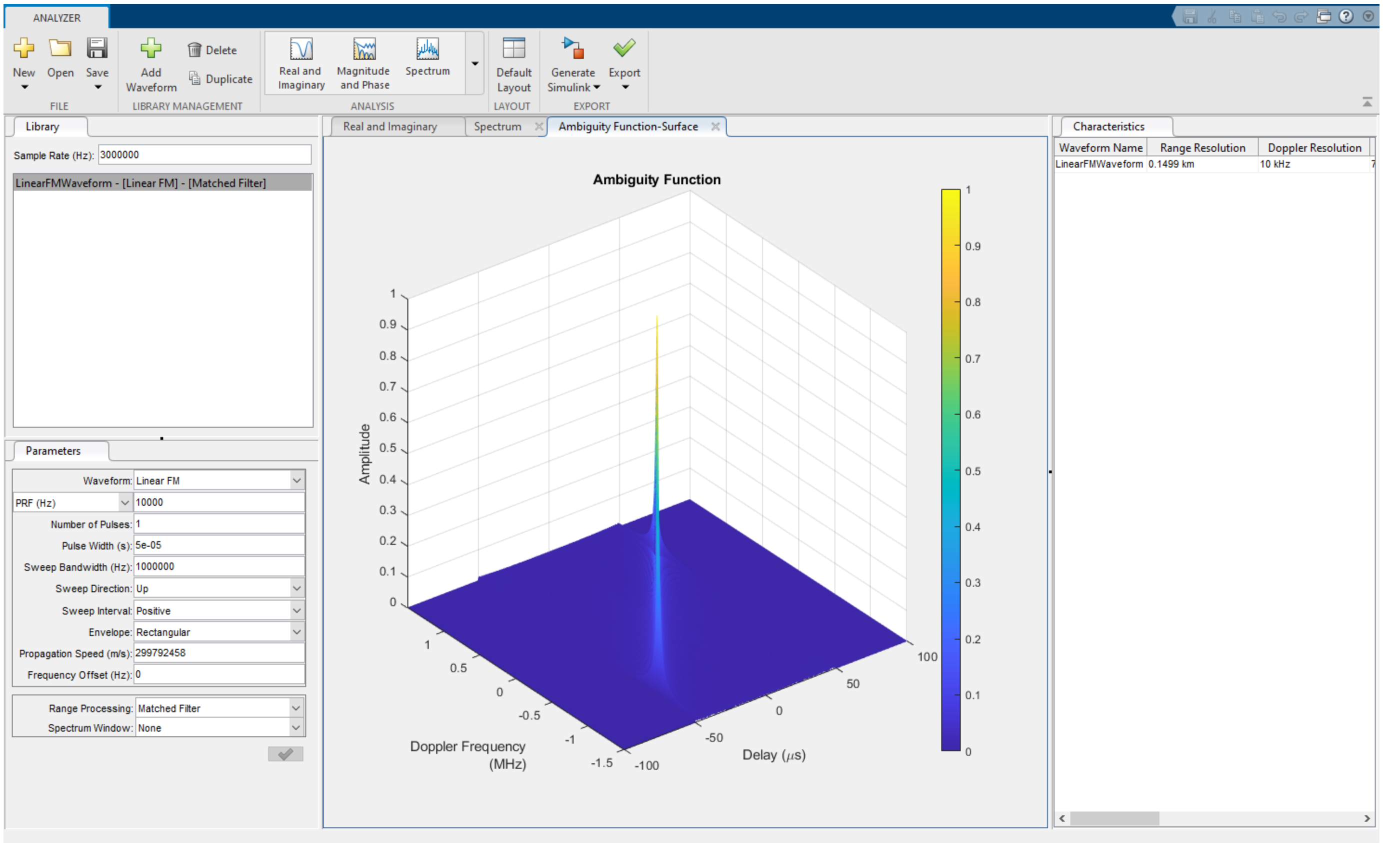
In the **Parameters** tab, change the **Waveform** to `Linear FM`. Then, change the waveform name to `LinearFMWaveform`. This type of pulse has a varying frequency, which can either increase or decrease as a linear function of time. Keep the sample rate at 3 MHz.

Choose the **Sweep Direction** as `Up`, and the **Sweep Bandwidth** as 1 MHz.

You can see that keeping the same pulse width as in the previous example, improves the range resolution to 150 m, as shown in the **Characteristics** tab.



While the range resolution gets better, the Doppler resolution is worse than the resolution of a rectangular waveform. You can see this by selecting the **Surface** ambiguity plot. The **Ambiguity Function-Surface** tab shows this tradeoff between Doppler resolution and range resolution.

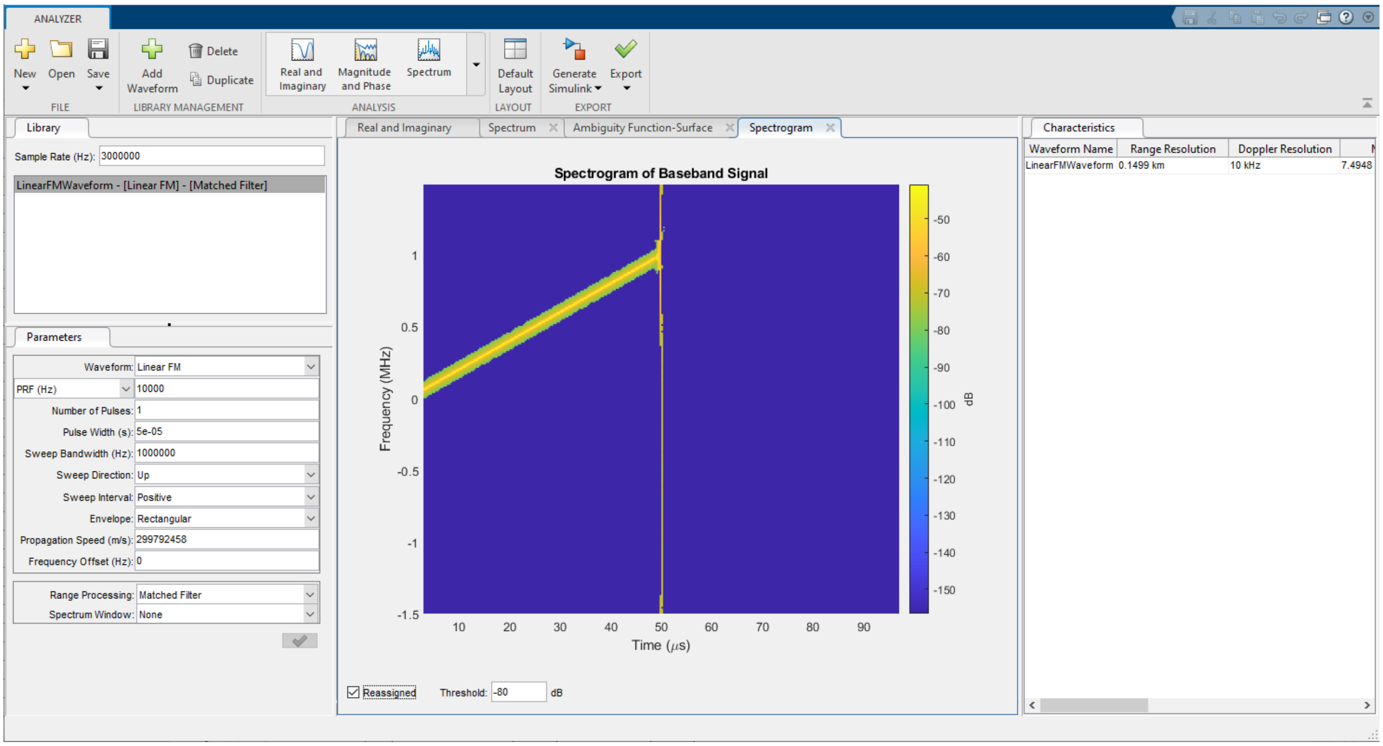


Linear FM Waveform Spectrogram

This example shows how to display the spectrogram of a linear FM waveform with and without frequency reassignment.

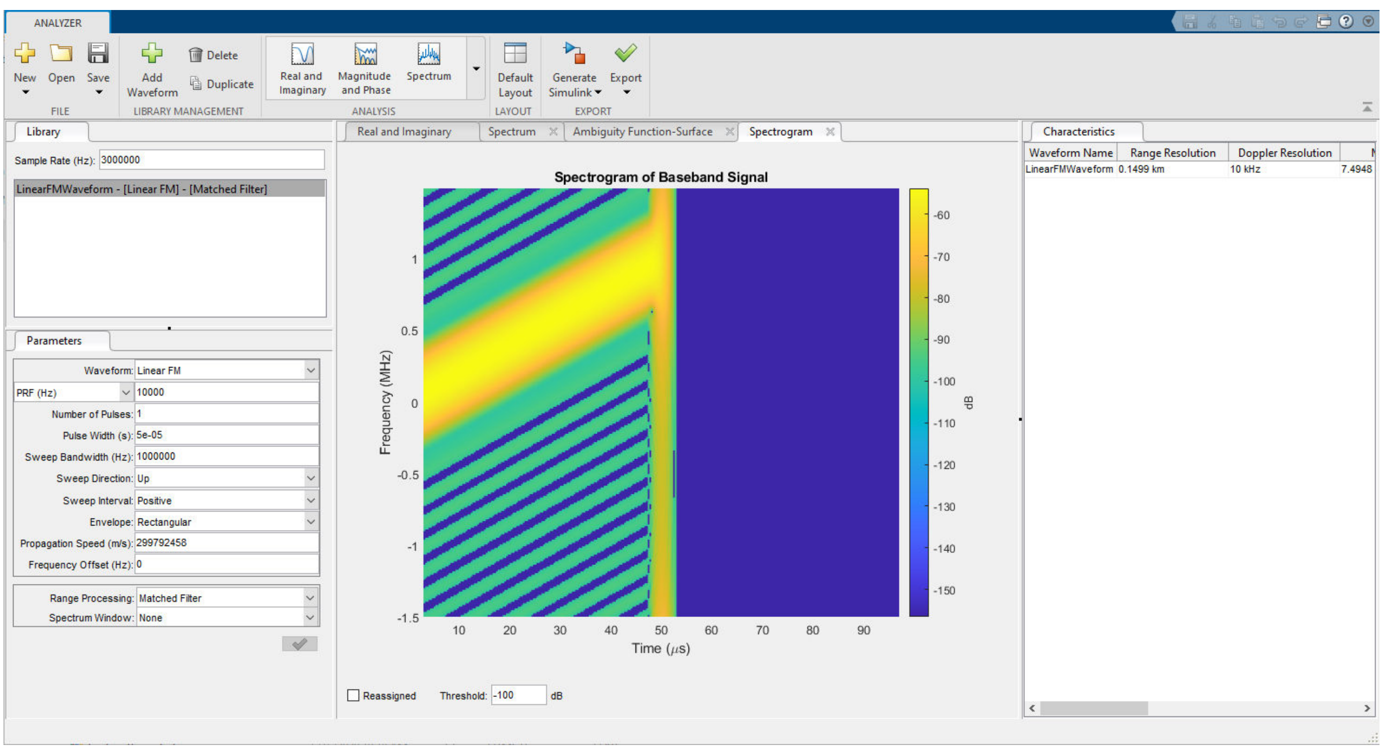
Use the same signal parameters as in the previous example.

Select **Spectrogram** from the **Signal Plots** drop-down menu. Then, select the **Reassigned** check box to show the frequency reassigned spectrogram (reassignment is turned on by default). Set the **Threshold** to -100 dB. Frequency reassignment is a technique for sharpening the magnitude spectrogram of a signal using information from its phase spectrum. For more information on frequency reassignment, see Fulop and Kelly (2006) [1].



You can vary the **Threshold** setting to show or hide weaker spectrum components.

To view the conventional spectrogram, clear the **Reassigned** check box.



Again, you can vary the **Threshold Value** setting to show or hide weaker spectrum components.

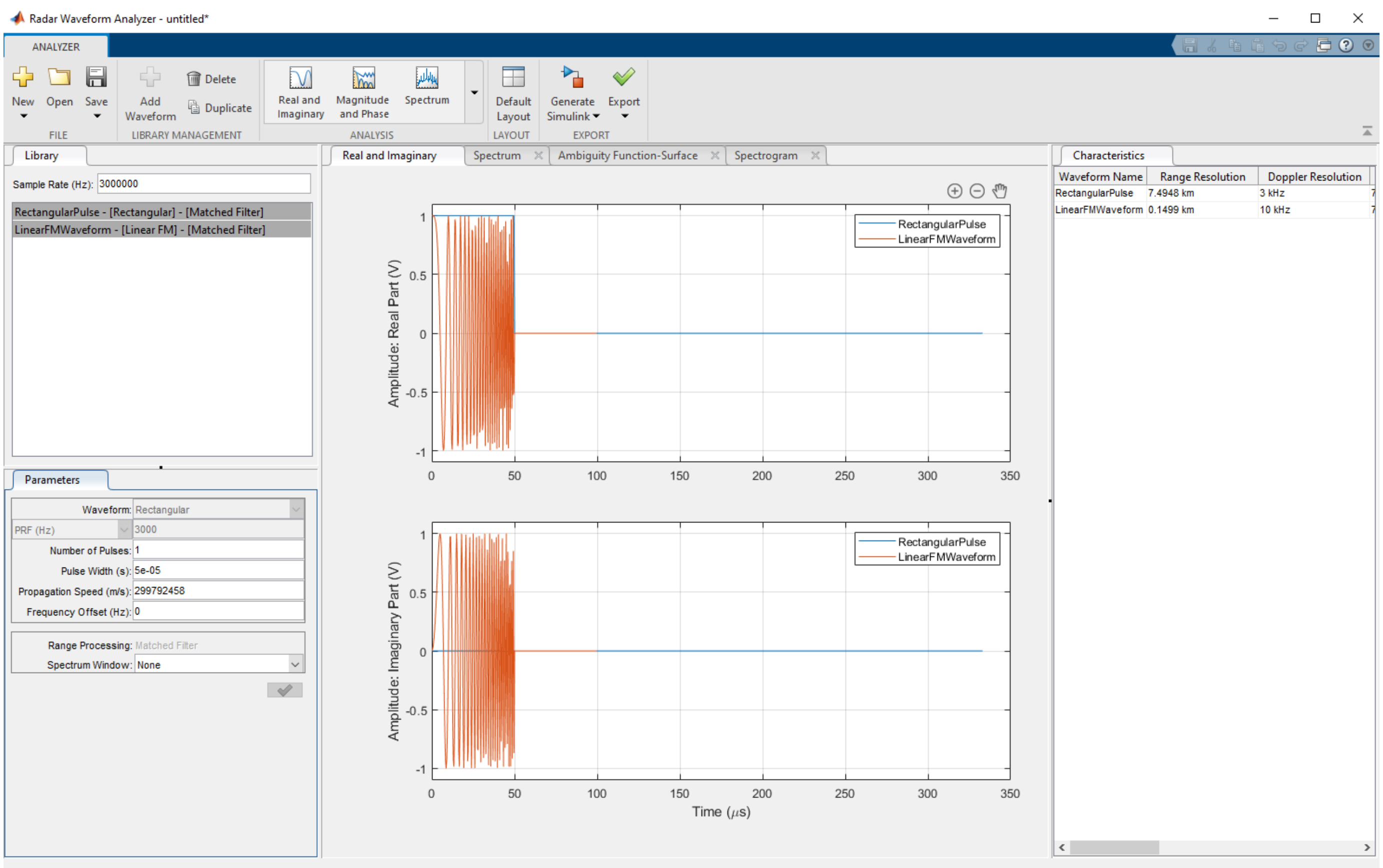
Display and Analyze Two Signals

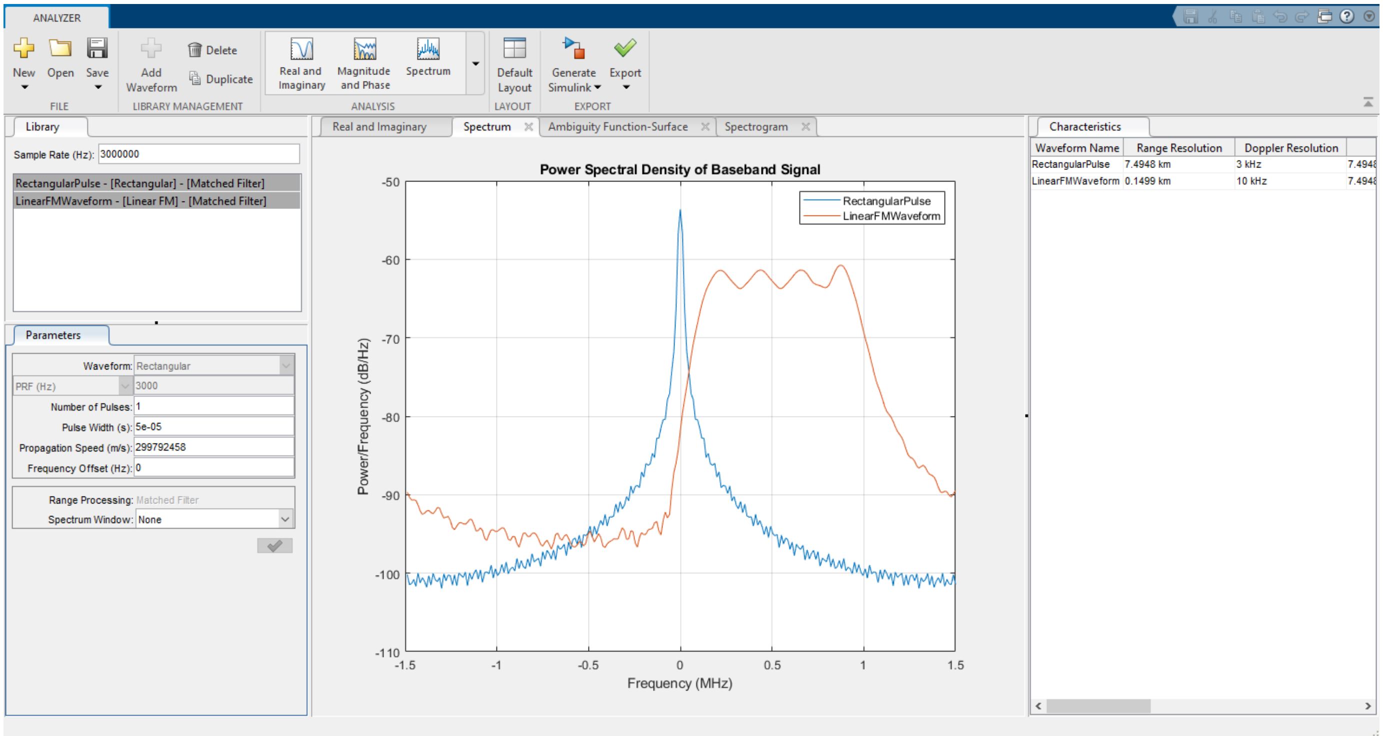
This example shows how to display the two signals simultaneously.

First, create a rectangular waveform with the same parameters as used in the first example. Then, rename the waveform to `RectangularPulse`.

Next, create an LFM waveform. Click the **Add Waveform** button. Rename the second waveform to `LinearFMPulse`. Set the waveform parameters to the same values as in the second example.

Select both waveforms in the **Library** panel using **Ctrl**+click. The display now shows the waveforms, spectra, and characteristics for both waveforms.





Programmatic Use

You can run **pulseWaveformAnalyzer** from the command line.

`pulseWaveformAnalyzer(wav)` opens the **Pulse Waveform Analyzer** app and imports and plots the waveform `wav`. `wav` can be a variable in the workspace representing a waveform object such as:

```
wav = phased.LinearFMWaveform(SampleRate=1e6, ...
    SweepBandwidth=200e3, ...
    PulseWidth=1e-3,PRF=1e3);
pulseWaveformAnalyzer(wav)
```

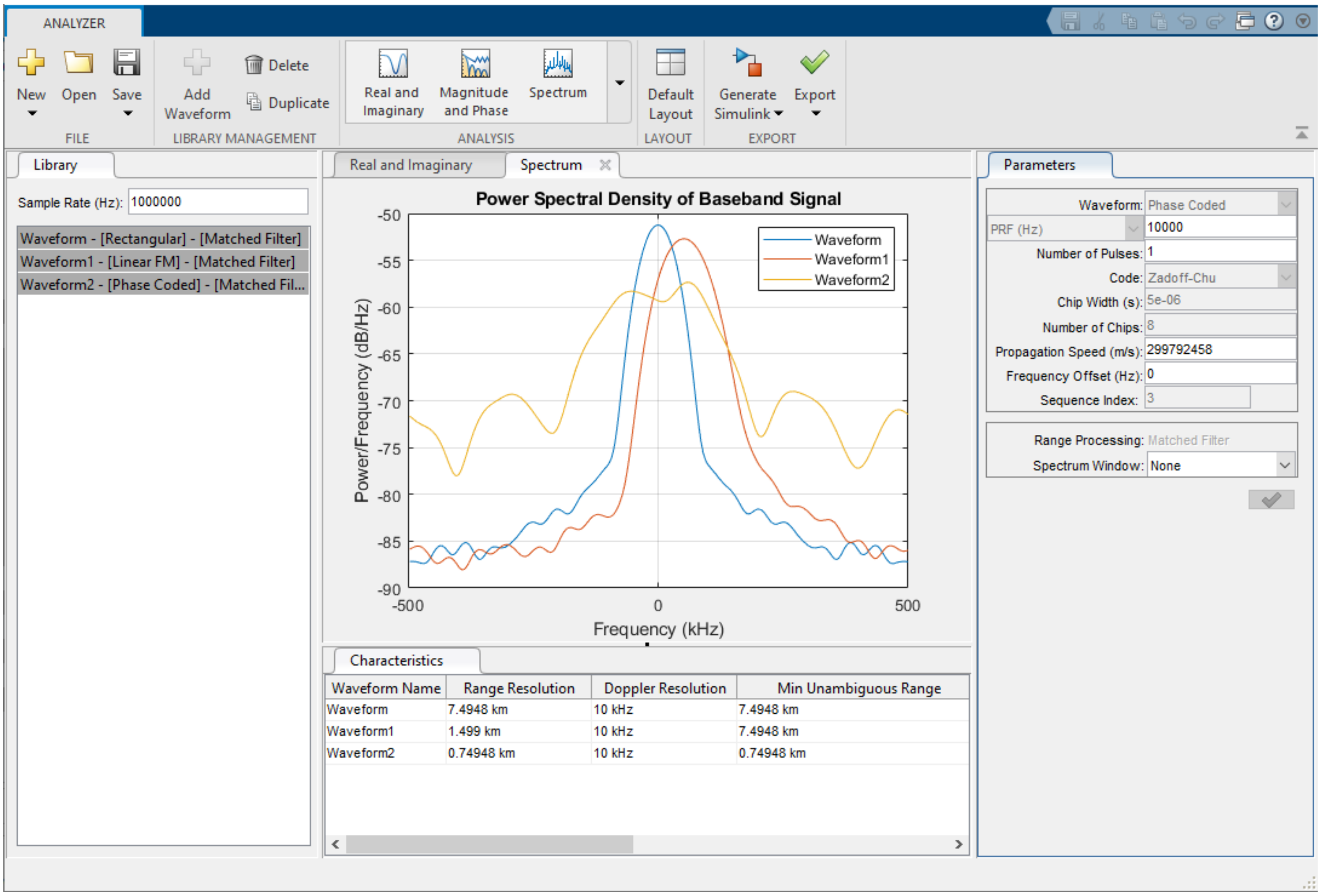
or you can enter the object directly:

```
pulseWaveformAnalyzer(phased.LinearFMWaveform( ...
    SampleRate=1e6, ...
    SweepBandwidth=200e3, ...
    PulseWidth=1e-3,PRF=1e3))
```

`pulseWaveformAnalyzer(wavlib)` opens the **Pulse Waveform Analyzer** app and imports a `pulseWaveformLibrary` object, `wavlib`. For example, construct the waveform library object from three waveforms with a common sample rate of 1 MHz. Then run from the command line:

```
waveform1 = {'Rectangular','PRF',1e4,'PulseWidth', 50e-6};
waveform2 = {'LinearFM','PRF',1e4,'PulseWidth',50e-6, ...
    'SweepBandwidth',1e5,'SweepDirection','Up',...
    'SweepInterval', 'Positive'};
waveform3 = {'PhaseCoded','PRF',1e4,'Code','Zadoff-Chu', ...
```

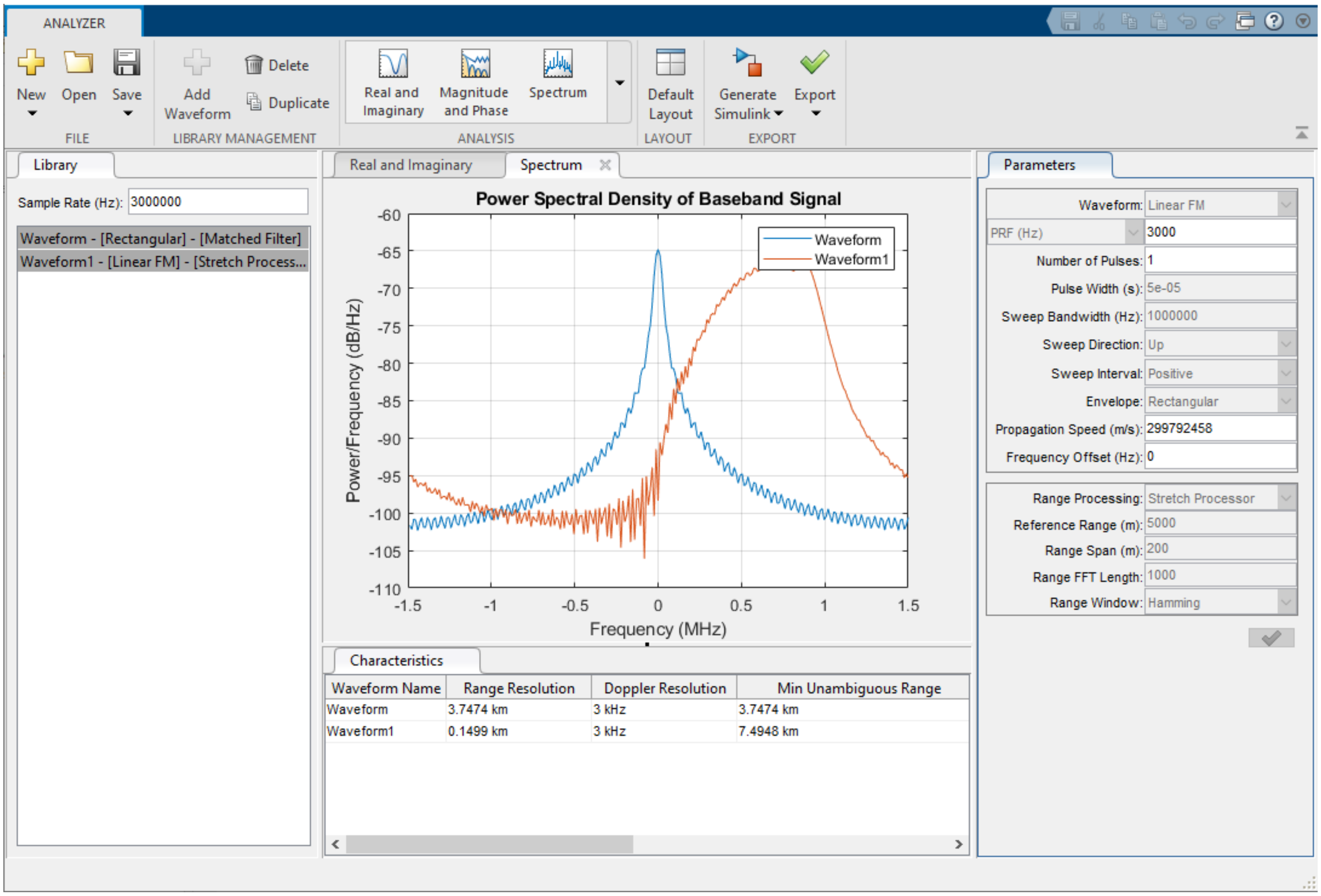
```
    'SequenceIndex',3,'ChipWidth',5e-6,'NumChips',8};
fs = 1e6;
wavlib = pulseWaveformLibrary('SampleRate',fs, ...
    'WaveformSpecification',{waveform1,waveform2,waveform3});
pulseWaveformAnalyzer(wavlib)
```



`pulseWaveformAnalyzer(comprlib)` opens the **Pulse Waveform Analyzer** app and imports a `pulseCompressionLibrary` object, `comprlib`. For example, using the waveforms from the "Rectangular Waveform" on page 4-4 and "Linear FM Waveform" on page 4-7 examples, create a matched filter for the rectangular waveform and a stretch processor for the linear FM waveform. Set the sample rate to 3 MHz, the pulse width of the rectangular wave to 25 µs, the pulse width of the linear wave to 50 µs, and the pulse repetition frequency to 3000 Hz. Export the compressed waveforms to the waveform app with these commands:

```
fs = 3e6;
rectpw = 25e-6;
linpw = 50e-6;
prf = 3e3;

waveform1 = {'Rectangular','PRF',prf,...
    'PulseWidth',rectpw};
waveform2 = {'LinearFM','PRF',prf,'PulseWidth',linpw,...
    'SweepBandwidth',1e6,'SweepDirection','Up',...
    'SweepInterval','Positive'};
```

```
procspec1 = {'MatchedFilter','SpectrumWindow','Hann'};
procspec2 = {'StretchProcessor','ReferenceRange',5000,...
    'RangeSpan',200,'RangeWindow','Hamming'};

comprlib = pulseCompressionLibrary(...
    'WaveformSpecification',{waveform1, waveform2},...
    'ProcessingSpecification',{procspec1, procspec2},...
    'SampleRate',fs,'PropagationSpeed',physconst('Lightspeed'));
pulseWaveformAnalyzer(comprlib)
```



Version History

Introduced in R2014b

References

[1] Fulop, Sean A., and Kelly Fitz. "Algorithms for Computing the Time-Corrected Instantaneous Frequency (Reassigned) Spectrogram, with Applications." *The Journal of the Acoustical Society of America* 119, no. 1 (January 2006): 360–71.

See Also

Apps
Sensor Array Analyzer

Sensor Array Analyzer

Analyze beam patterns and performance characteristics of linear, planar, 3-D, and arbitrary sensor arrays

Description

The **Sensor Array Analyzer** app enables you to construct and analyze common sensor array configurations. These configurations range from 1-D to 3-D arrays of antennas, sonar transducers, and microphones, and can contain subarrays. After you specify array and sensor parameters, the app displays basic performance characteristics such as array directivity and array dimensions. You can then create various directivity plots and images.

Array Types

You can use this app to show the directivity of these arrays:

2D Arrays

- Uniform Linear Array (ULA)
- Uniform Rectangular Array (URA)
- Uniform Circular Array (UCA)
- Uniform Hexagonal Array (UHA)
- Circular Planar Array
- Concentric Array

3D Arrays

- Spherical Array
- Cylindrical Array
- Arbitrary Array

Subarrays

You can use this app to create and analyze arrays containing subarrays to:

- Replicate an array along a spatial grid.
- Partition a larger array into subarrays.

Element Types

These elements are available to populate an array:

Non-Polarized Antennas

- Cardioid Antenna
- Cosine Antenna
- Custom Antenna

- Gaussian Antenna
- Isotropic Antenna
- Sinc Antenna

Polarized Antennas

- Crossed Dipole Antenna
- Custom Antenna
- NR Antenna
- Short Dipole Antenna

Microphones

- Cardioid Microphone
- Custom Microphone
- Omnidirectional Microphone

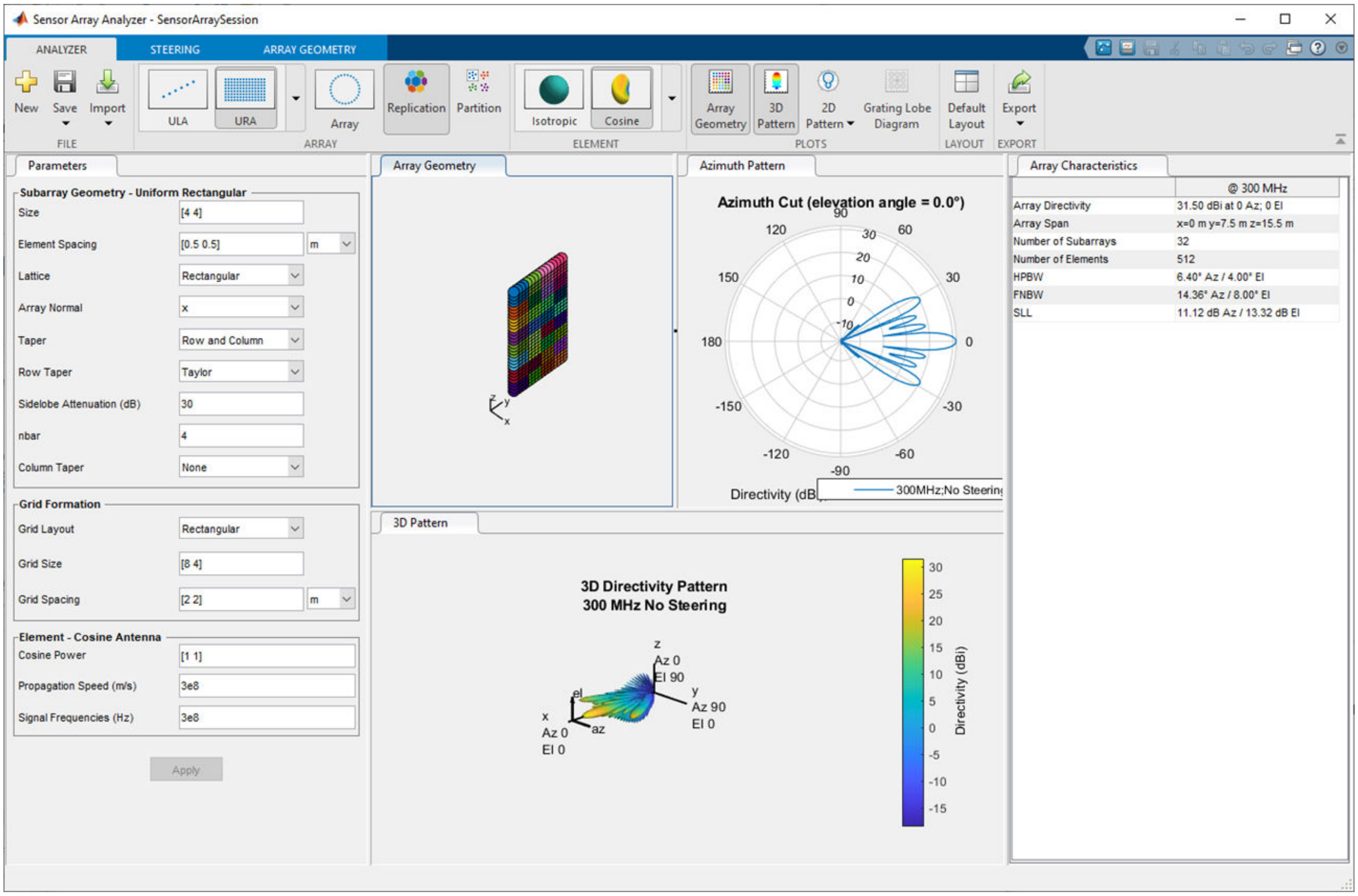
Sonar Transducers

- Isotropic Hydrophone
- Isotropic Projector

Plot Options

The **Sensor Array Analyzer** app can create these types of plots:

- Array Geometry
- 2-D Array Patterns
- 3-D Array Pattern
- Grating Lobes



Open the Sensor Array Analyzer App

- MATLAB toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command prompt: Enter `sensorArrayAnalyzer`.

Examples

Uniform Linear Array (ULA)

This example analyzes a 10-element uniform linear array (ULA) in a sonar application. The array consists of isotropic hydrophones. Design the array for a 10 KHz signal.

A uniform linear array has sensor elements that are equally spaced along a line.

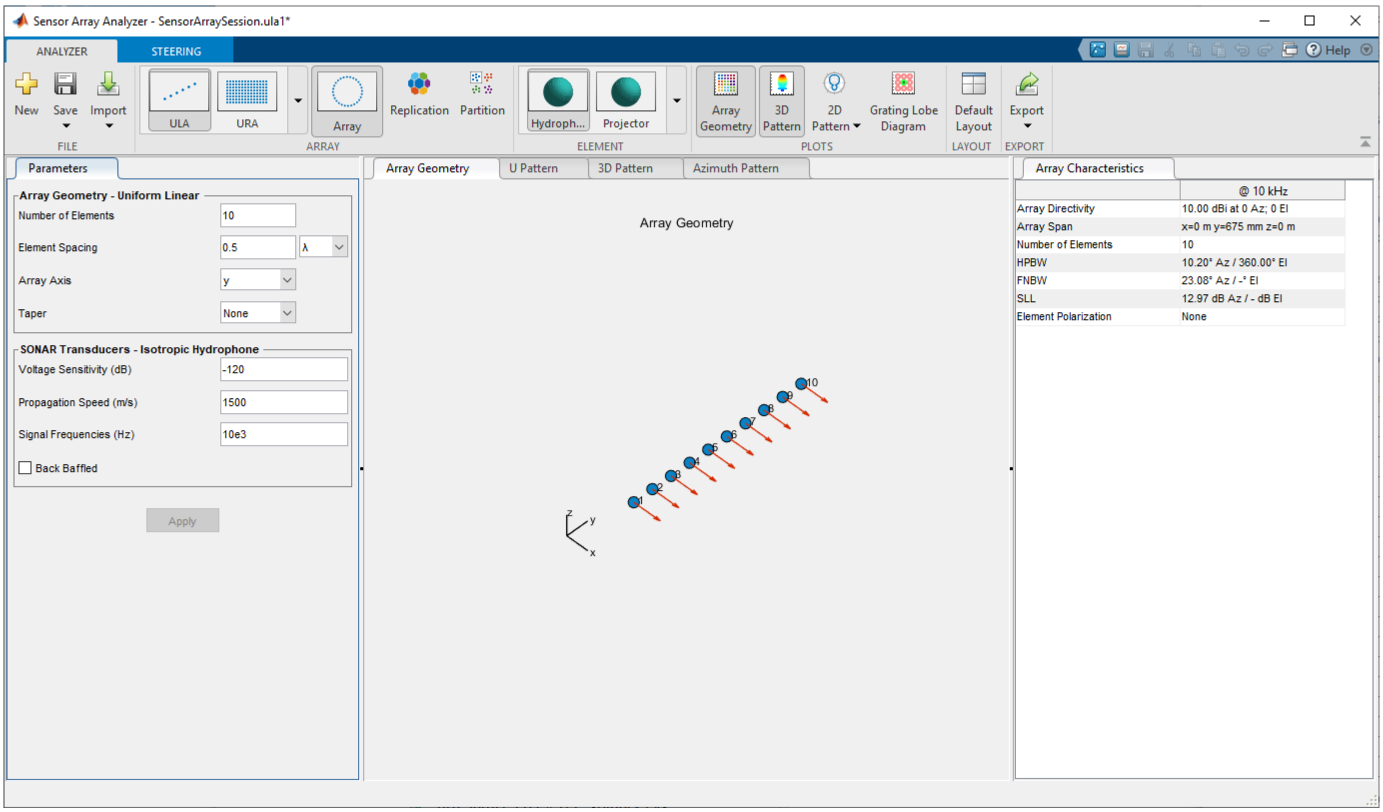
Under the **Analyzer** tab, in the **Array** section of the toolstrip, select **ULA**. In the **Element** section of the toolstrip, select **Hydrophone**.

Select the Parameters tab and set the **Number of Elements** to `10`. Set the **Element Spacing** to `0.5` wavelengths.

Design the array for a 10 KHz signal by setting **Signal Frequencies (Hz)** to `10000`. Then click the **Apply** button. You can change many menu items and apply the changes at any time. The parameters that appear in this tab depend on your choice of array and element.

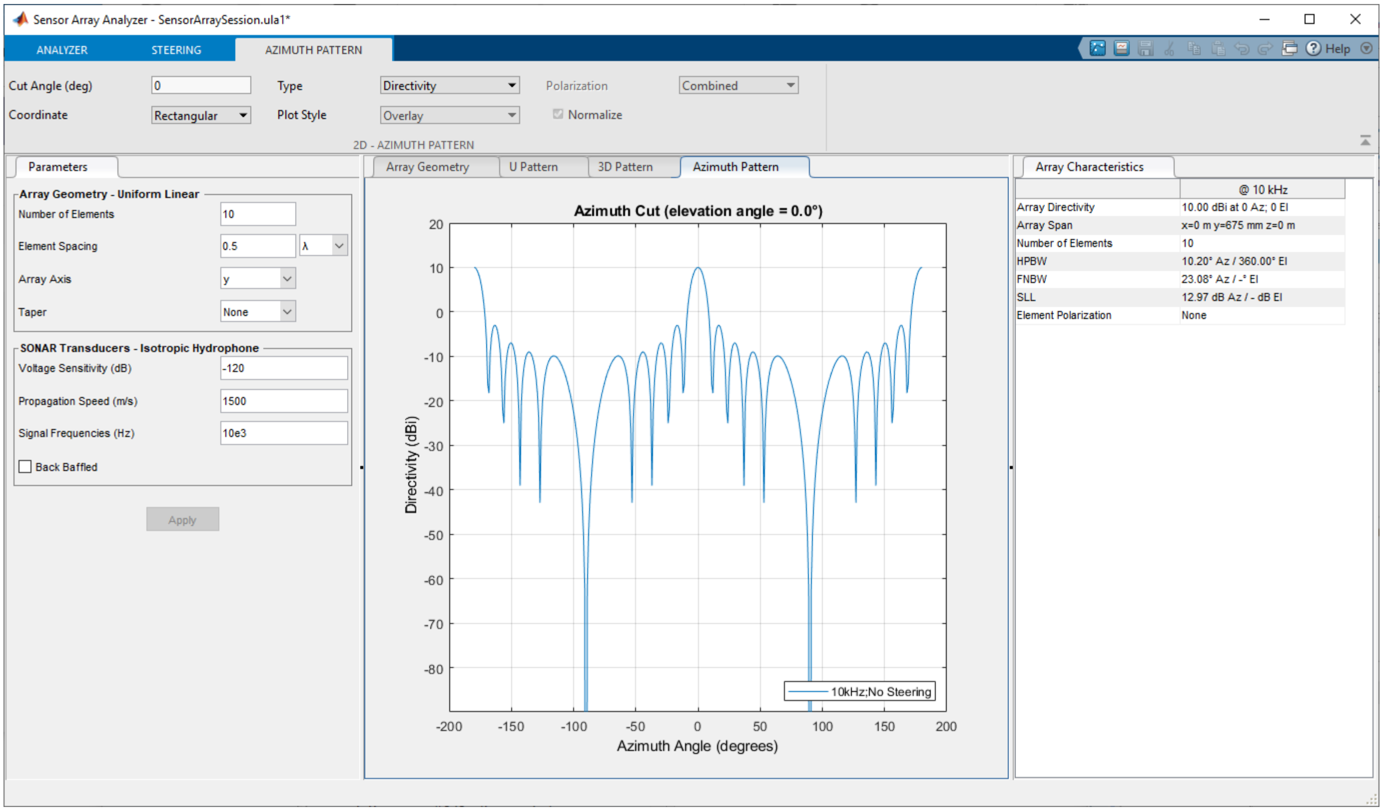
When you choose a sonar element, the app automatically sets the signal propagation speed in water to `1500`. You can set the signal propagation speed to any value by setting the **Propagation Speed (m/s)**.

Select the **Array Geometry** tab and use the check boxes to display element normals (**Show Normals**), element indices (**Show Index**), and element tapers (**Show Tapers**).



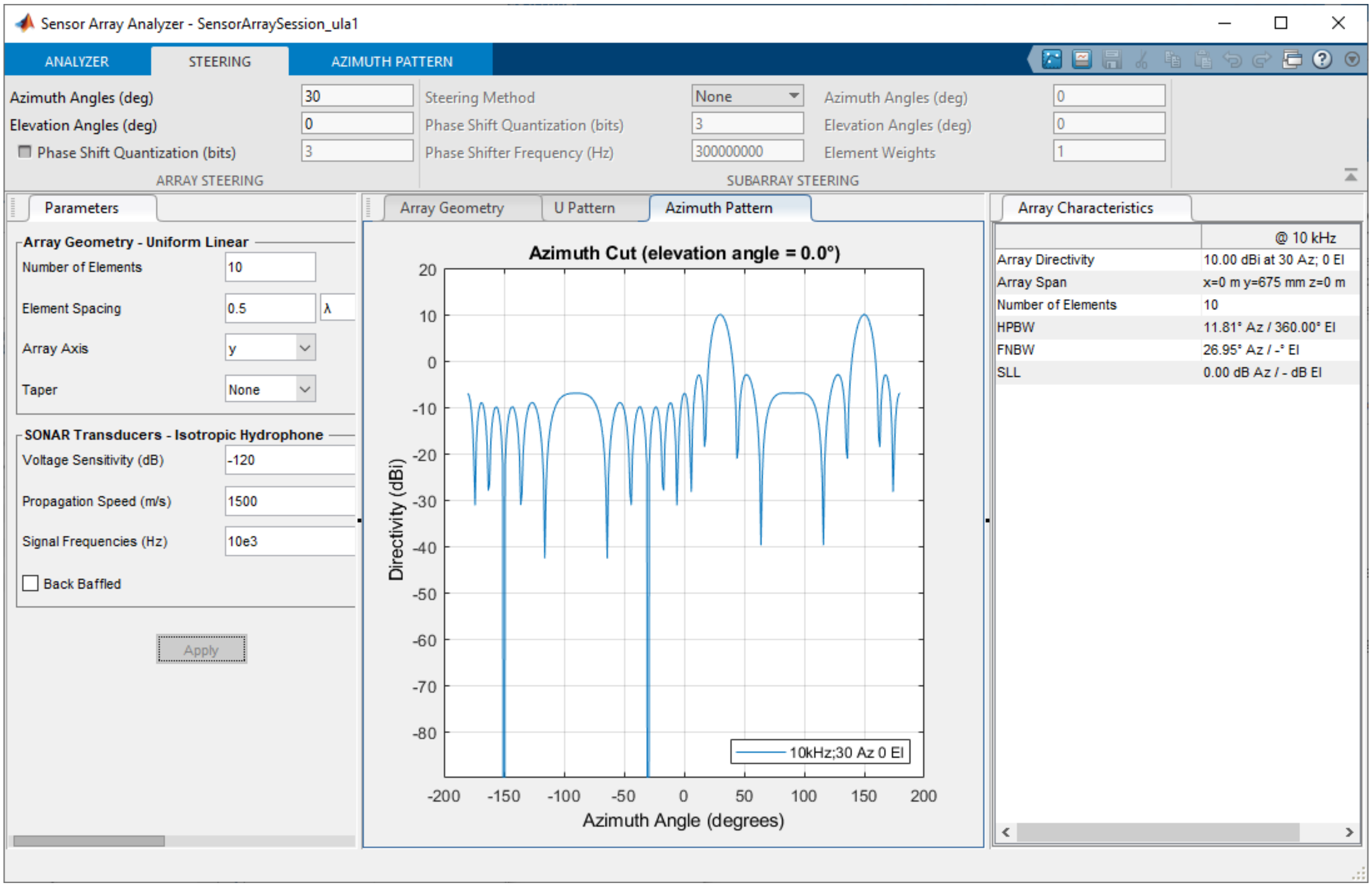
In the rightmost **Array Characteristics** panel, you can view the array directivity, half-power beam width (HPBW), first-null beam-width (FNBW), and side lobe level (SLL).

To display a directivity plot, go to the **Plots** section of the **Analyzer** tab. Select **Azimuth Pattern** from the **2D Pattern** menu. The azimuth directivity pattern is now displayed in the center panel of the app. Select the **Azimuth Pattern** tab, and set the **Coordinate** to `Rectangular`.



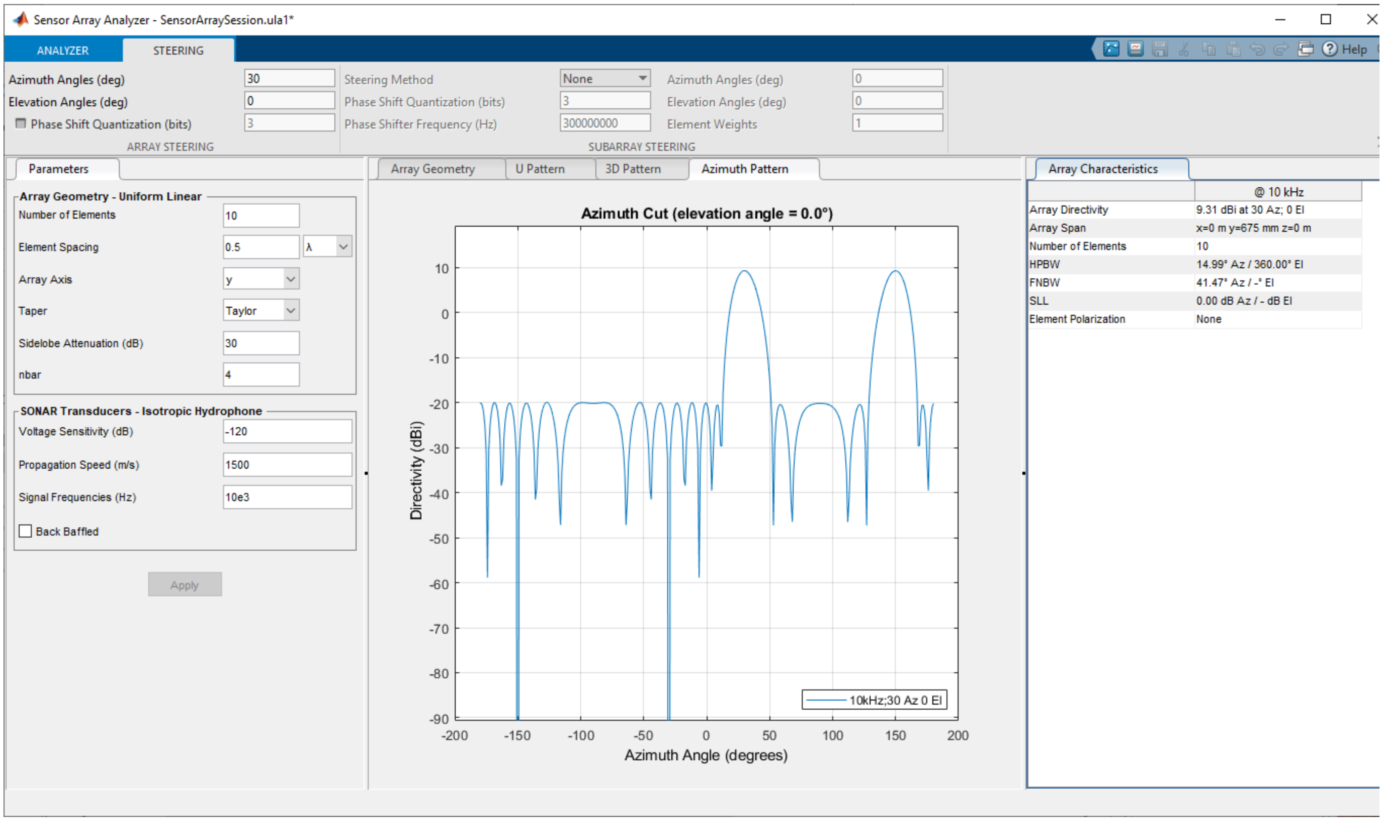
You can see the main lobe of the array directivity function (also called the main beam) at 0° and another main lobe at ±180°. Two main lobes appear because of the cylindrical symmetry of the ULA array.

A beam scanner works by successively pointing the array main lobe in different directions. In the **Steering** tab, set **Azimuth Angles (deg)** to `30` and **Elevation Angles (deg)** to `0`. This steers the main lobe to 30° in azimuth and 0° elevation.



One disadvantage of a ULA is its large side lobes. An examination of the array directivity shows two side lobes close to each main lobe, each down by about only 13 dB. A strong side lobe inhibits the ability of the array to detect a weaker signal in the presence of a larger nearby signal. By using array tapering, you can reduce the side lobes.

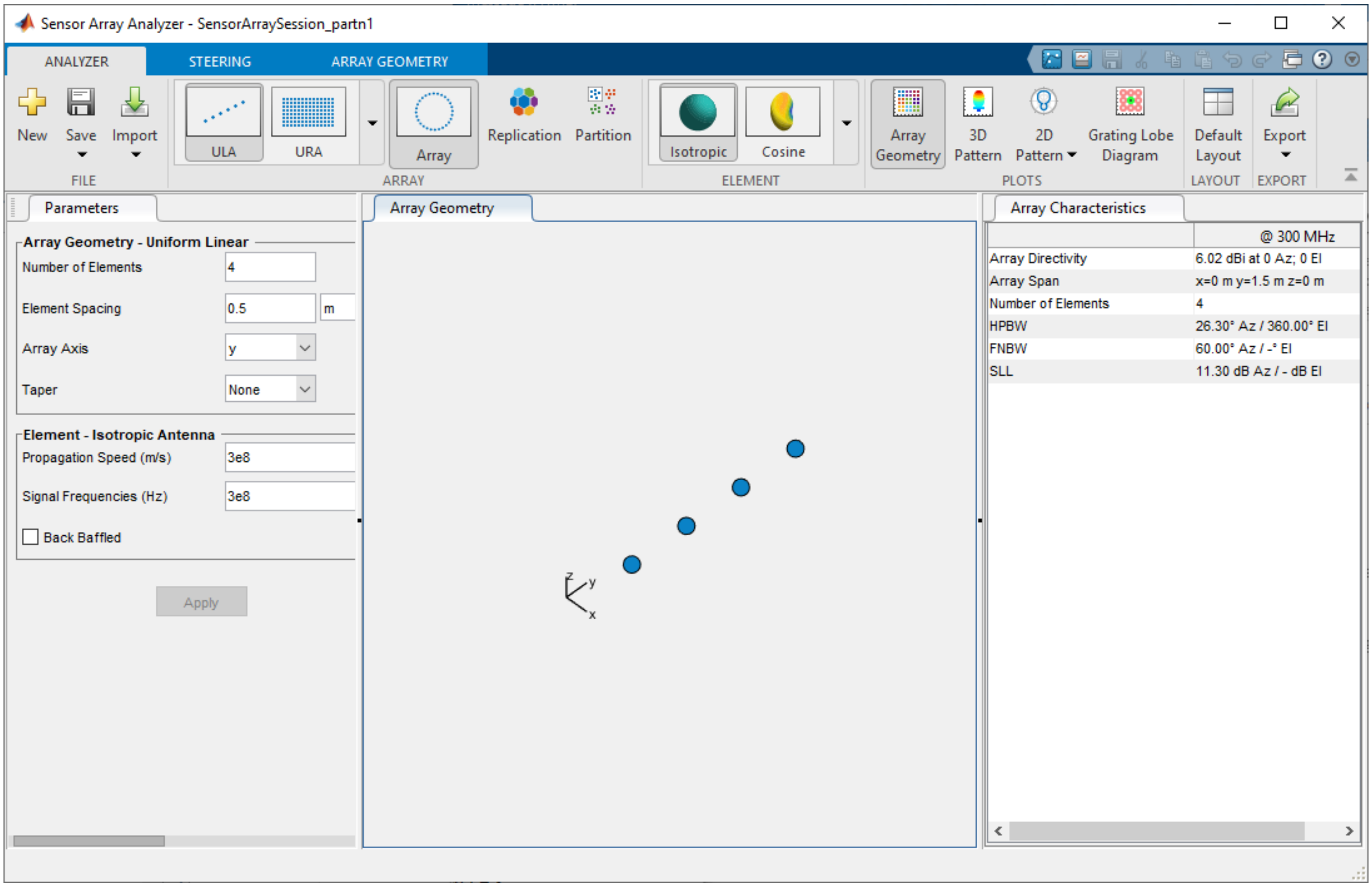
Use the **Taper** option to specify the array taper as a `Taylor` window with **Sidelobe Attenuation** set to `30` dB and **nbar** set to `4`. Click the **Apply** button.



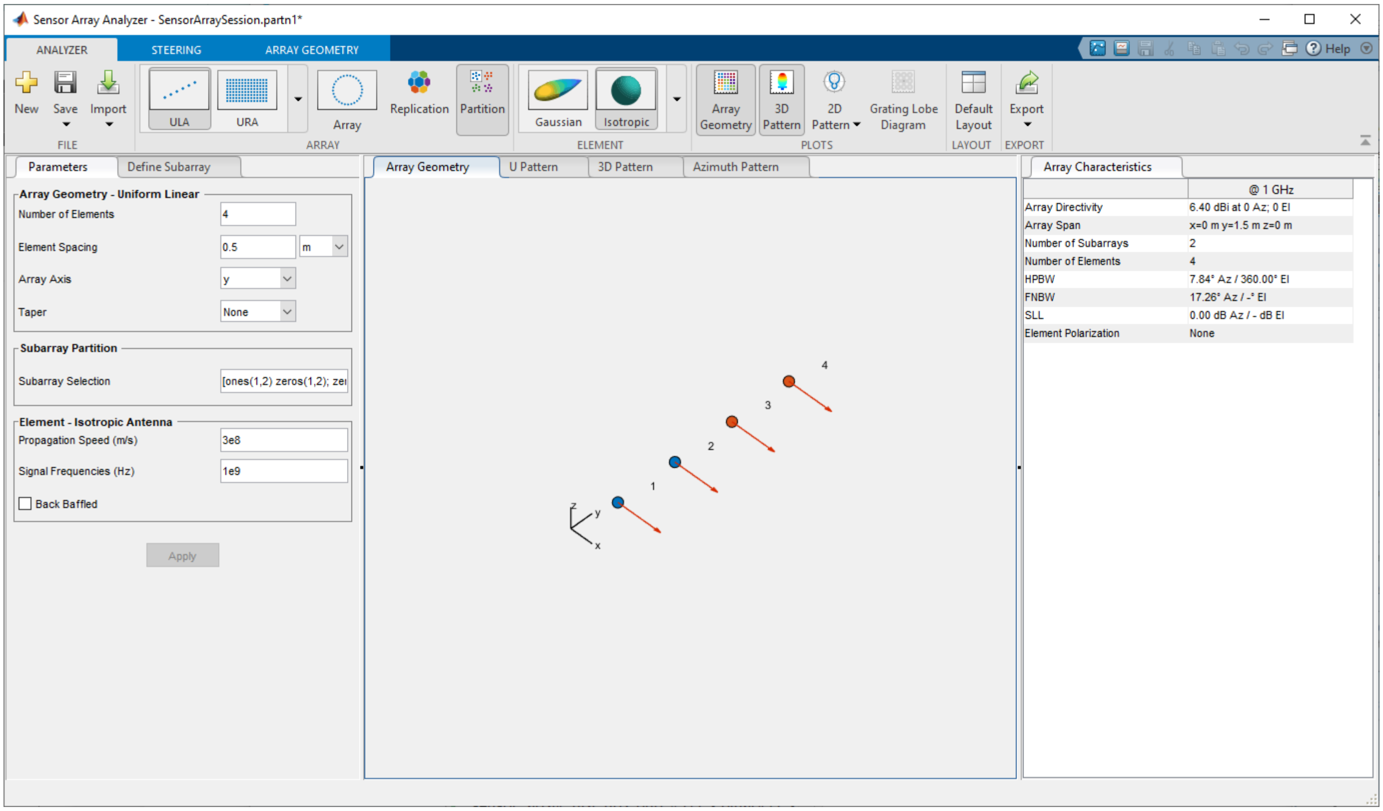
Azimuth Response of Partitioned ULA

This example plots the azimuth response of a four-element ULA partitioned into two two-element ULAs.

Under the **Analyzer** tab, in the **Array** section of the toolstrip, select **ULA**. Create a ULA with default parameters (with the number of elements set to 4 and the element spacing set to 0.5 meters).

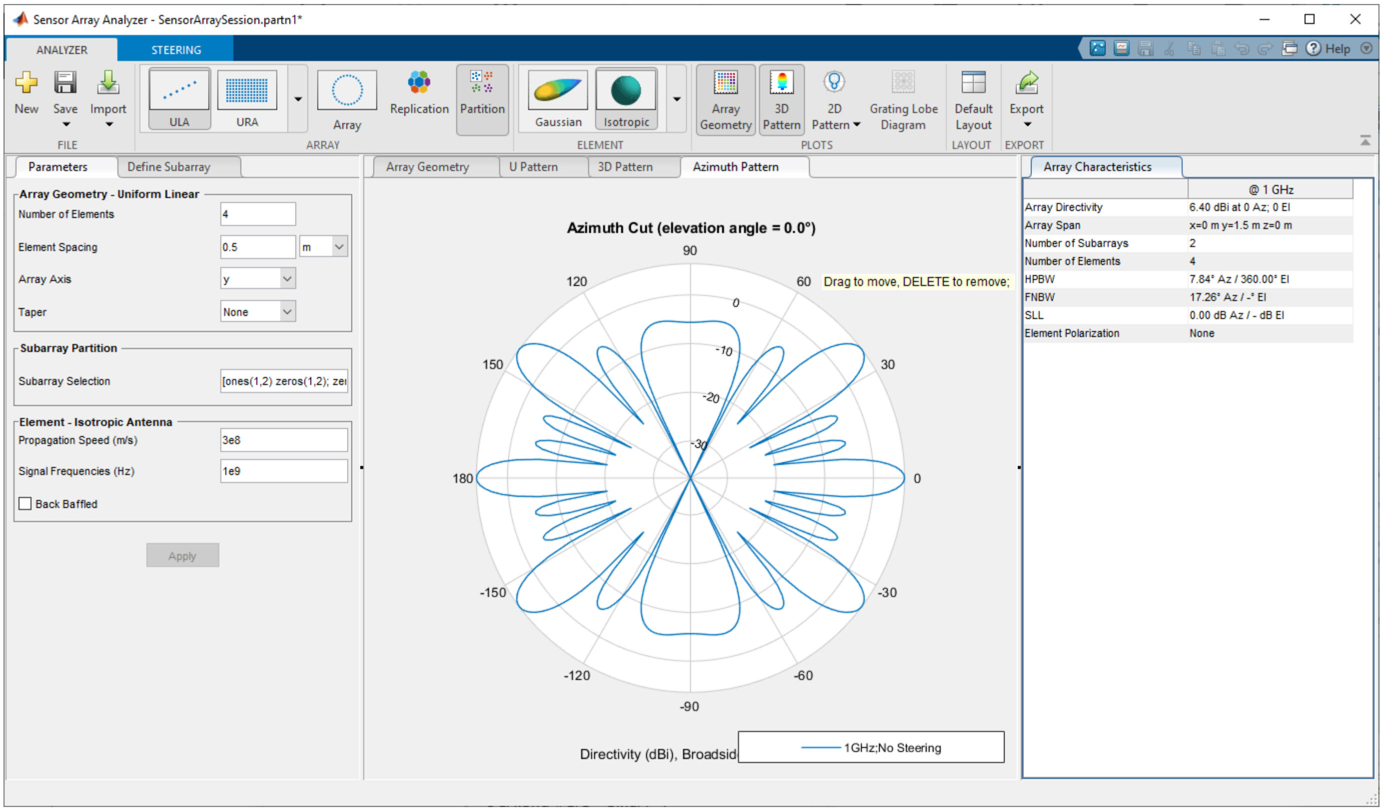


Select the **Partition** button on the **Analyzer**. Design the array for a 1 GHz signal by setting **Signal Frequencies (Hz)** to `1e9`. Then click the **Apply** button. You can change many menu items and apply the changes at any time. The parameters that appear in this tab depend on your choice of array and element.



The **subarray selection** menu item should read `[ones(1,2) zeros(1,2); zeros(1,2) ones(1,2)]`.

Select **2D Pattern** in the **Analyzer** tab and choose the `Azimuth pattern` to visualize the 2-D azimuth pattern in polar coordinates.



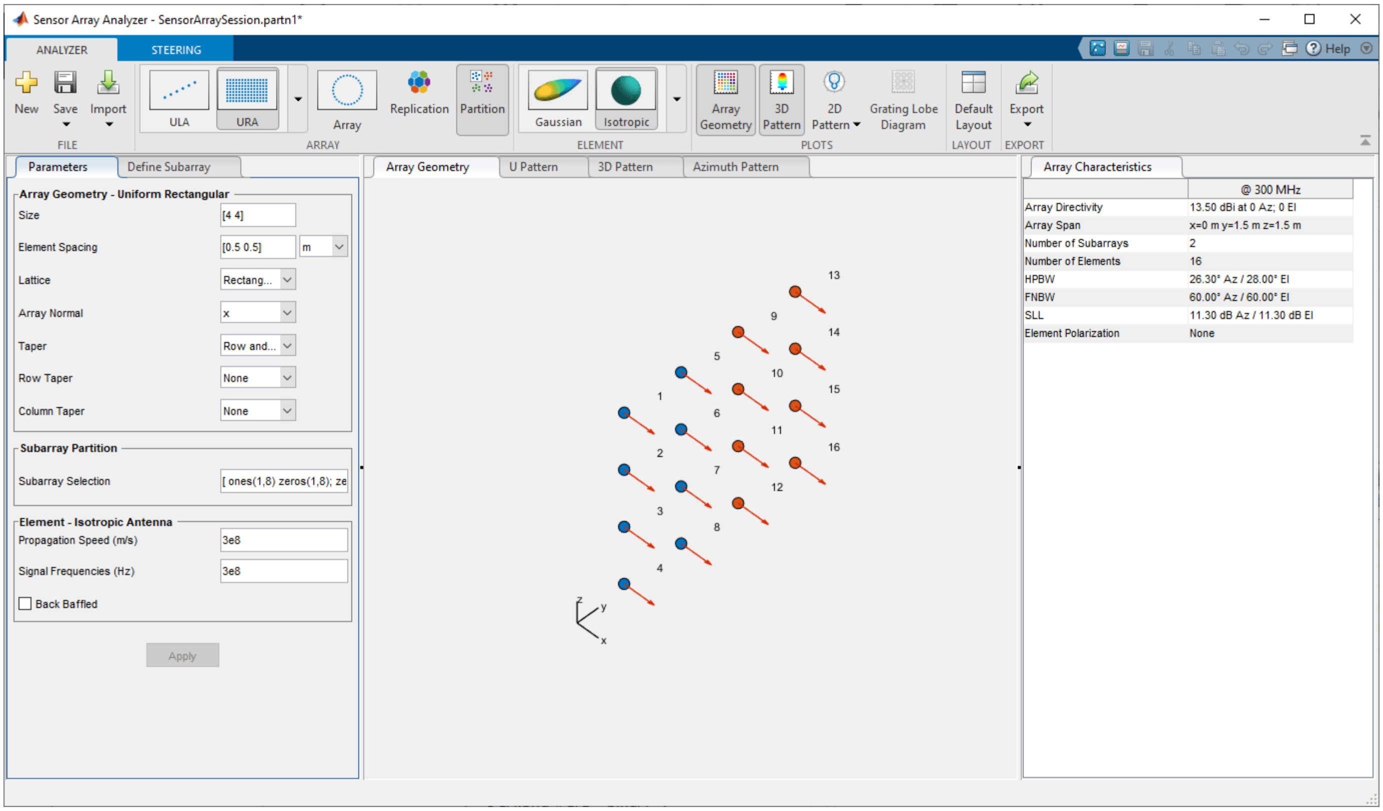
Re-Partition URA

A partitioned array consists of multiple subarrays in which each array element can be assigned to one or more subarrays. After creating the partition array, you can then re-assign elements to different subarrays. For example, create a 4-by-4 uniform rectangular array (URA) containing 16 elements. Selecting the **Partition** tab converts the URA into a 4-by-4 partitioned array with subarrays indicated by different colors. The partitioning is controlled by the **Subarray Selection** matrix.

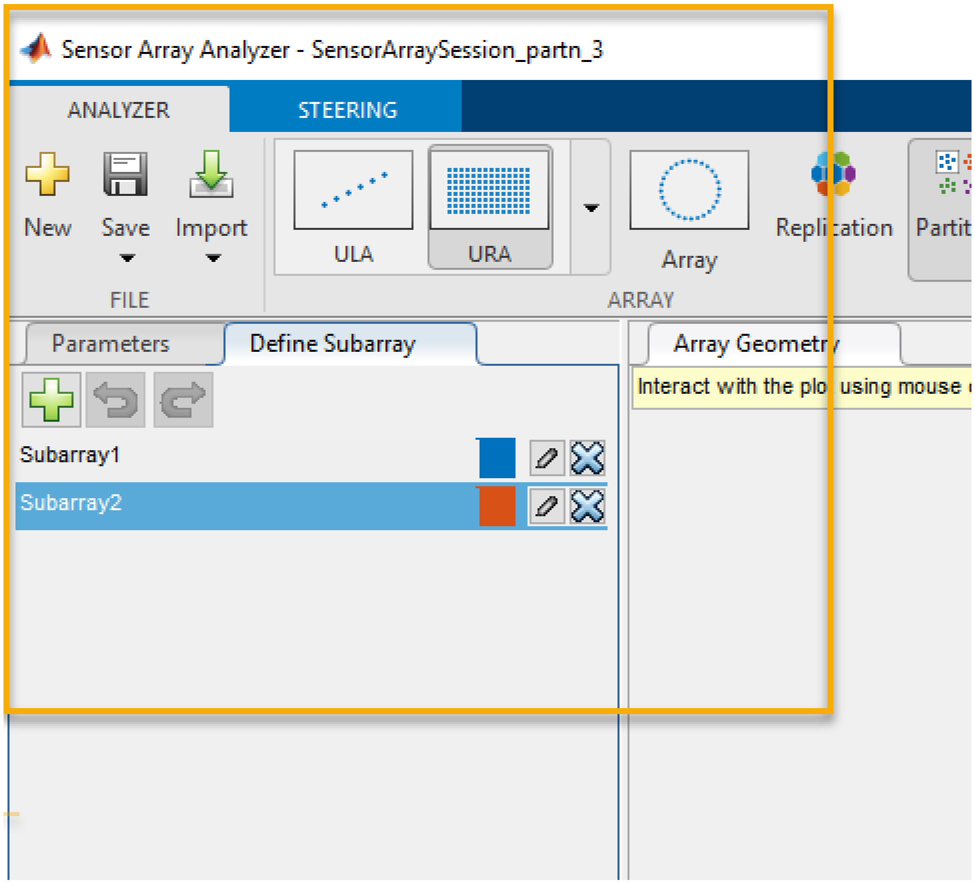
```
[ ones(1,8) zeros(1,8); zeros(1,8) ones(1,8)]
```

The default subarray selection matrix assigns each element to one subarray. In this matrix, the number of columns equals the number of array elements. Each row corresponds to a subarray. This 2-by-16 matrix assigns elements 1 – 8 to subarray 1 and elements 9 –16 to subarray 2.

To re-partition an array, you can edit the **Subarray Selection** matrix. Select the **Define Subarray** tab to rearrange the elements belonging to the subarrays.



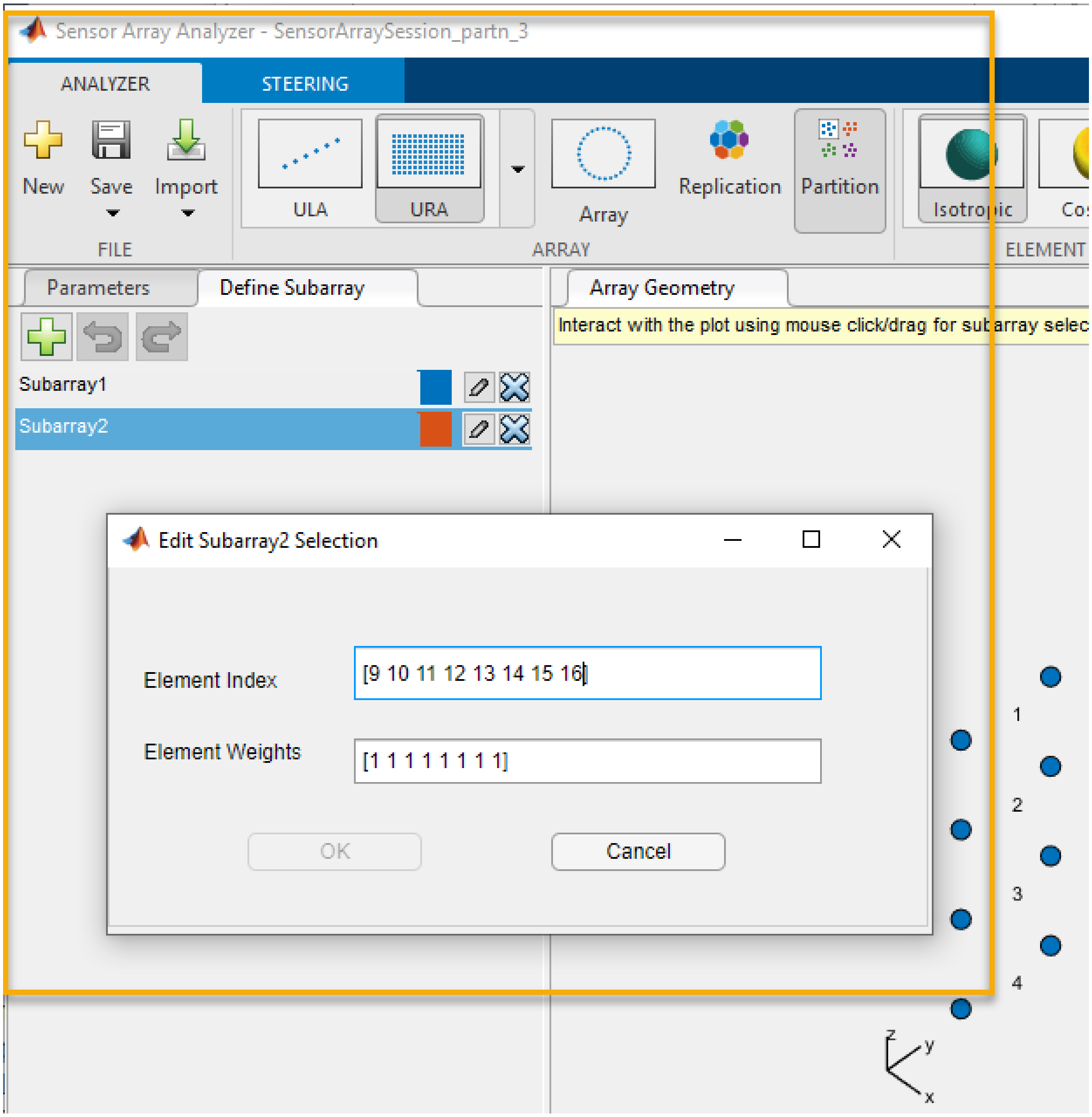
Selecting the Define Subarray tab brings up the subarray editor.



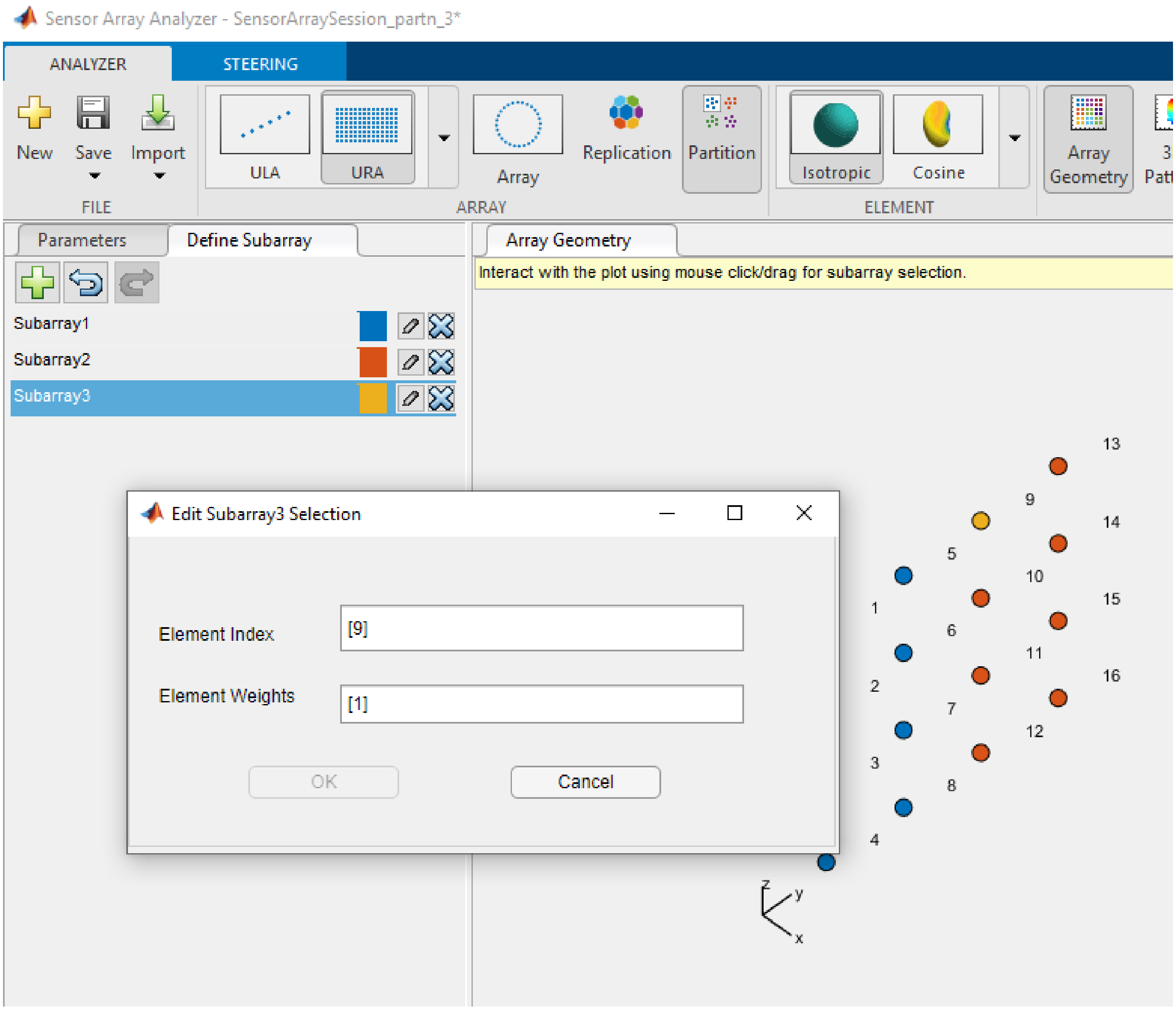
You can:

- Select the pencil icon next to Subarray1 to edit the elements and weights in subarray 1.
- Select the pencil icon next to Subarray2 to edit the elements and weights in subarray 2.
- Select the green cross icon at the top to create an empty subarray.

Select **Subarray 2** to display the element indices belonging to **Subarray 2**.



Remove element 9 and its weight. Select the green cross add a new subarray, **Subarray 3**. Then add element 9 to the new subarray.



The new subarray and its added element appears in yellow.

Uniform Rectangular Array (URA)

This example shows how to construct a 6-by-6 uniform rectangular array (URA) designed to detect and localize a 100 MHz signal.

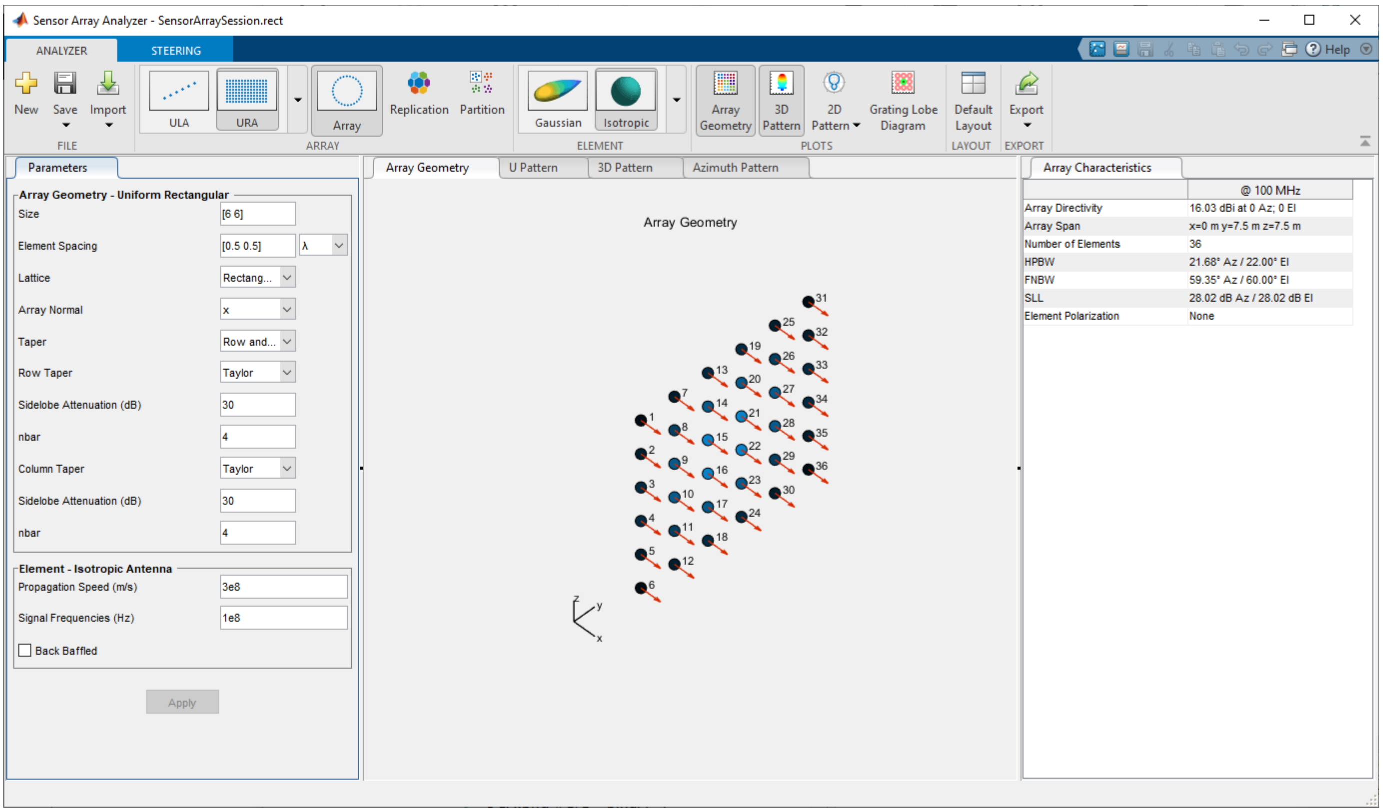
Under the **Analyzer** tab, in the **Array** section of the toolstrip, select **URA**. In the **Element** section of the toolstrip, select **Isotropic**.

Design the array for a 100 MHz signal by setting **Signal Frequencies** to `100e6` and the row and column **Element Spacing** to `[0.5 0.5]` wavelength.

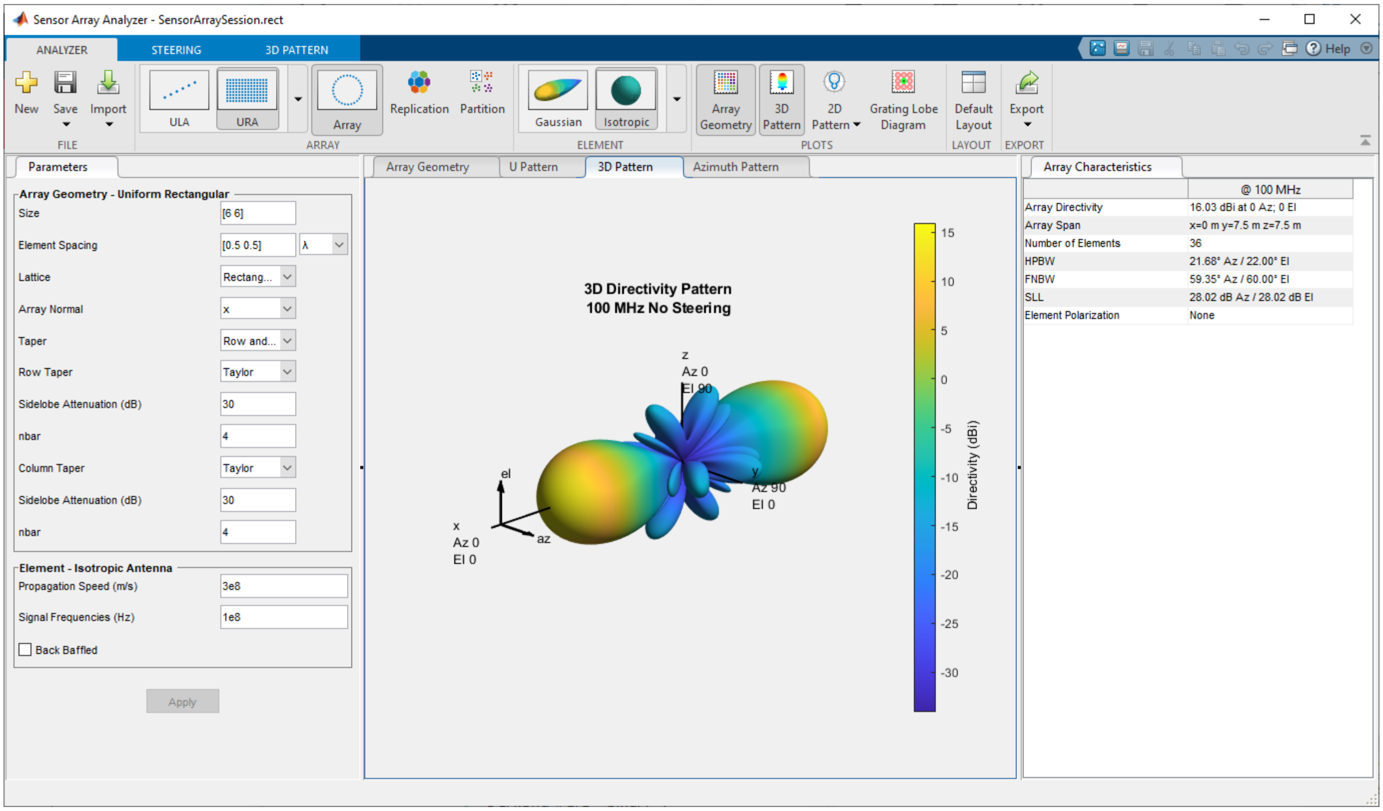
Select the Parameters tab and set the **Size** to `[6,6]`.

From the **Taper** drop-down, choose `Row and Column`. Set **Row Taper** and **Column Taper** to a `Taylor` window using default taper parameters. Click the **Apply** button to apply the changes. You can change many menu items and apply the changes at any time. The parameters that appear in this tab depend on your choice of array and element.

The shape of the array is shown in this figure.



Next, display a 3-D array pattern by selecting **3D Pattern** in the **Plots** section of the **Analyzer** tab.



A significant performance measure for any array is directivity. You can use the app to examine the effects of tapering on array directivity. Without tapering, the array directivity for this URA is 17.16 dB. With tapering, the array directivity is reduced to 16.03 dBi.

Grating Lobes for a Rectangular Array

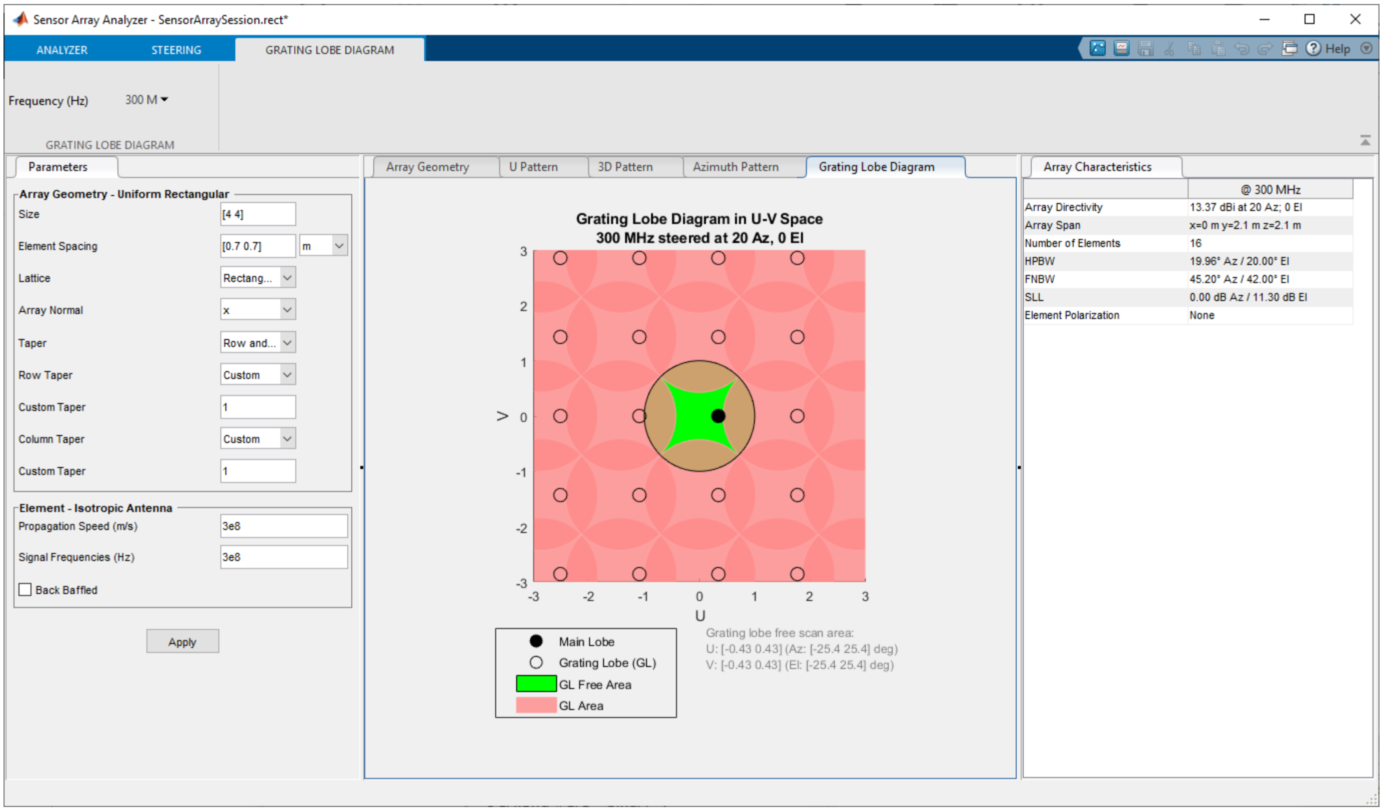
This example shows the grating lobe diagram of a 4-by-4 uniform rectangular array (URA) designed to detect and localize a 300 MHz signal.

Under the **Analyzer** tab, in the **Array** section of the toolstrip, select **URA**. In the **Element** section of the toolstrip, select **Isotropic**. Set the **Size** to `[4,4]`. In the **Steering** tab, set **Azimuth Angles (deg)** to `20` and **Elevation Angles (deg)** to `0`.

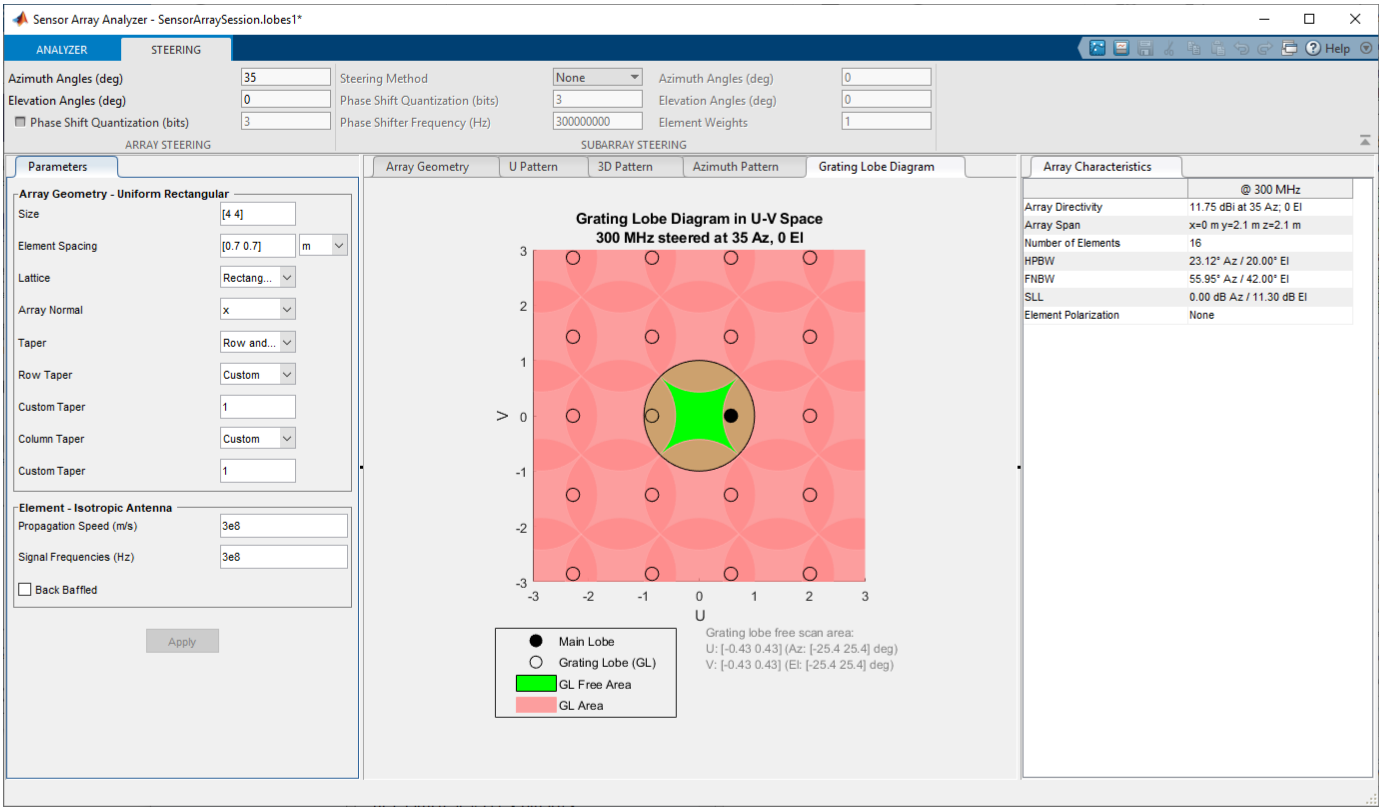
Design the array for a 300 MHz signal by setting **Signal Frequencies** to `3e8` and the row and column **Element Spacing** to `[0.7,0.7]` wavelength. By setting the row and column **Element Spacing** to `[0.7,0.7]` wavelengths, you create a spatially under-sampled, array. Then click the **Apply** button.

Select **Grating Lobe Diagram** from the **Plots** section to plot the grating lobes.

This figure shows the grating lobe diagram produced when you beamform the array towards the angle [20,0]. The main lobe is designated by the small black-filled circle. The multiple grating lobes are designated by the small unfilled black circles. The larger black circle is called the physical region, for which $u^2 + v^2 \leq 1$. The main lobe always lies in the physical region. The grating lobes can sometimes lie outside the physical region. Any grating lobe in the physical region leads to an ambiguity in the direction of the incoming wave. The green region shows where the main lobe can be pointed without any grating lobes appearing in the physical region. If the main lobe is set to point outside the green region, a grating lobe can move into the physical region.

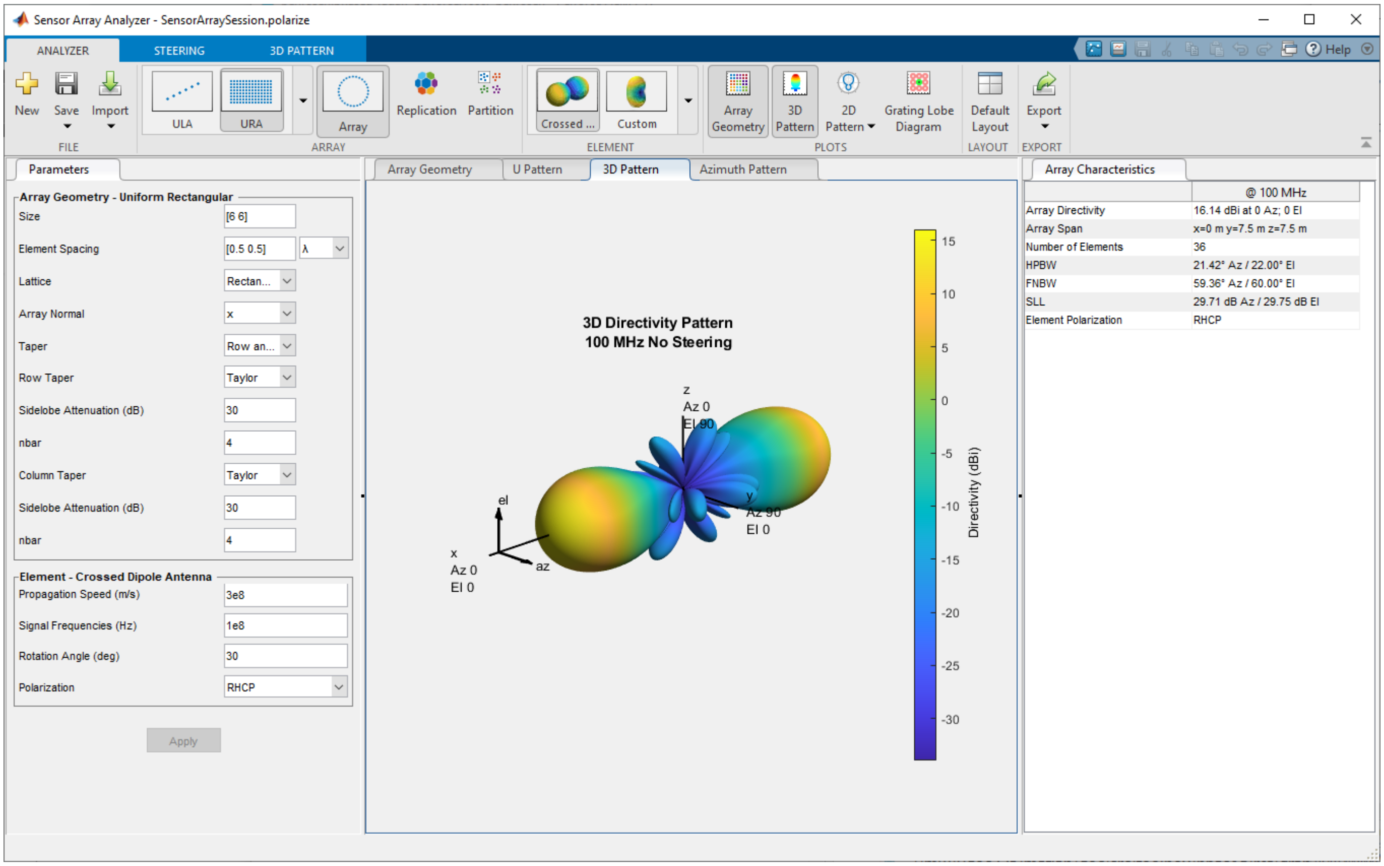


The next figure shows what happens when the pointing direction lies outside the green region. In the **Steering** tab, set **Azimuth Angles (deg)** to 35 and **Elevation Angles (deg)** to 0. In this case, one grating lobe moves into the physical region.



URA Containing Polarized Elements

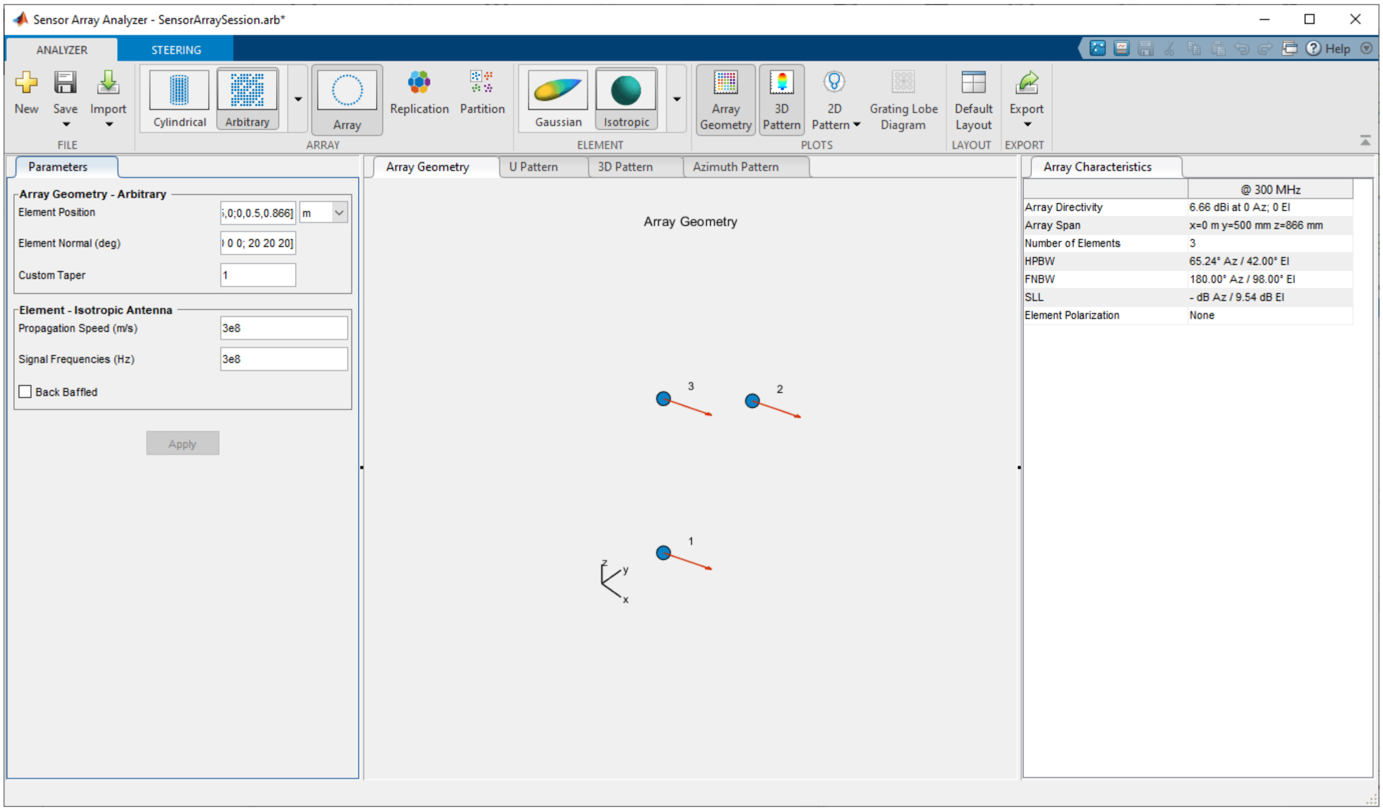
1. Under the **Analyzer** tab, in the **Array** section of the toolstrip, select **URA**. In the **Element** section of the toolstrip, select **Crossed Dipole**. Select RHCP as the **Polarization** and set the **Rotation Angle** to 30°.
2. Design the array for a 100 MHz signal by setting **Signal Frequencies** to `100e6` and the row and column **Element Spacing** to `[0.5 0.5]` wavelength.
3. Select the Parameters tab and set the **Size** to `[6,6]`.
4. From the **Taper** drop-down, choose `Row and Column`. Set **Row Taper** and **Column Taper** to a `Taylor` window using default taper parameters. Click the **Apply** button to apply the changes.



Specify Arbitrary Array Geometry

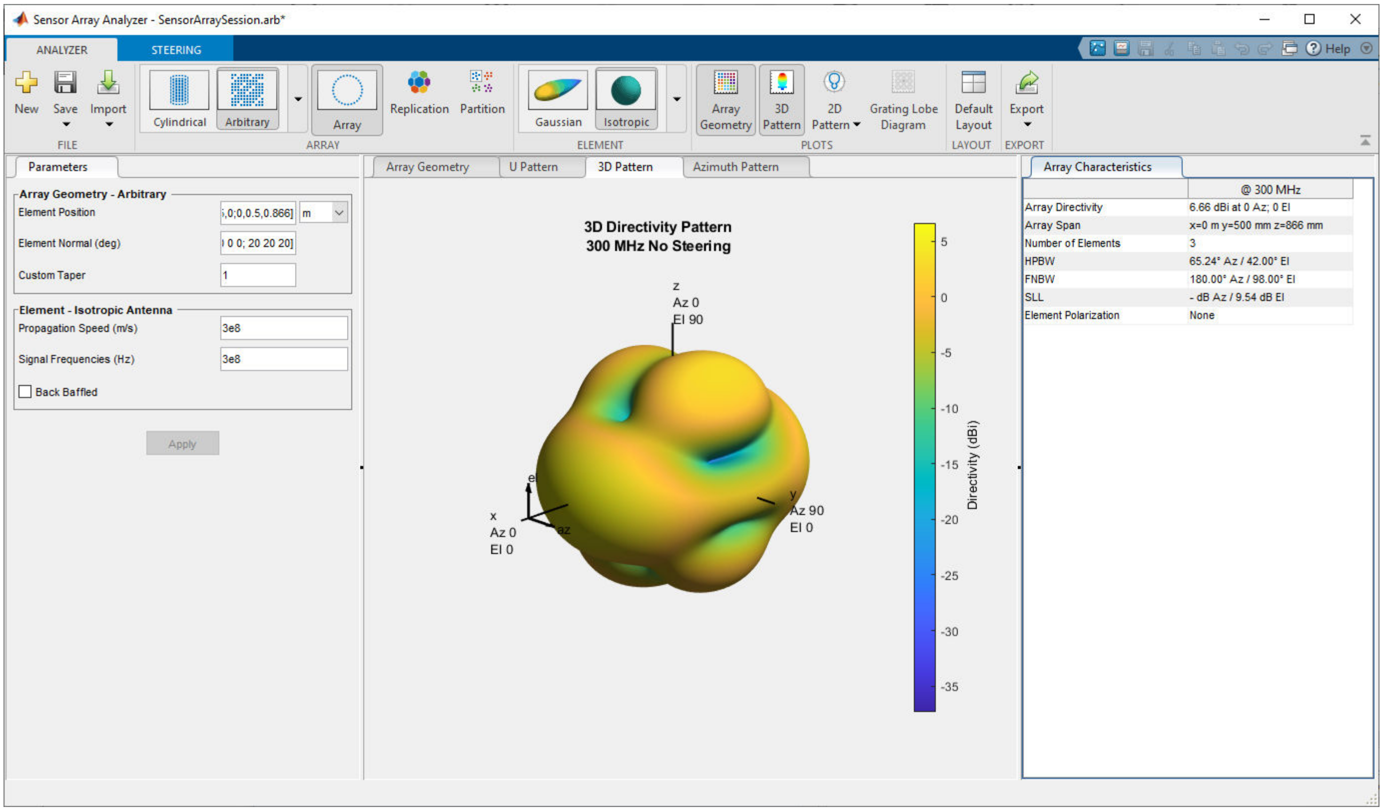
This example shows how to construct a triangular array of three isotropic antenna elements.

You can specify an array which has an arbitrary placement of sensors. Select **Arbitrary** in the **Array** drop-down. Select **Isotropic** from the **Element** menu. Enter the elements positions in the **Element Position** field. The positions of the three elements are `[0,0,0;0,0.5,0;0,0.5,0.866]`. All elements have the same normal direction, pointing to 0° azimuth and 20° elevation and to set the normal in the **Element Normal (deg)** type `[0 0 0; 20 20 20]` and click the **Apply** button. Select **Array Geometry** from the **Plots** section.



To show the 3-D array directivity, select **3D Pattern** from the **Plots** tab.

- You can use the **Orientation** dialog box to change the orientation of the array.
- The **Show Array** check box toggles off and on the display of the array.
- The **Show Local Coordinates** check box toggles off and on the display of the local coordinate system.
- The **Show Colorbar** check box toggles off and on the colorbar showing field intensities.



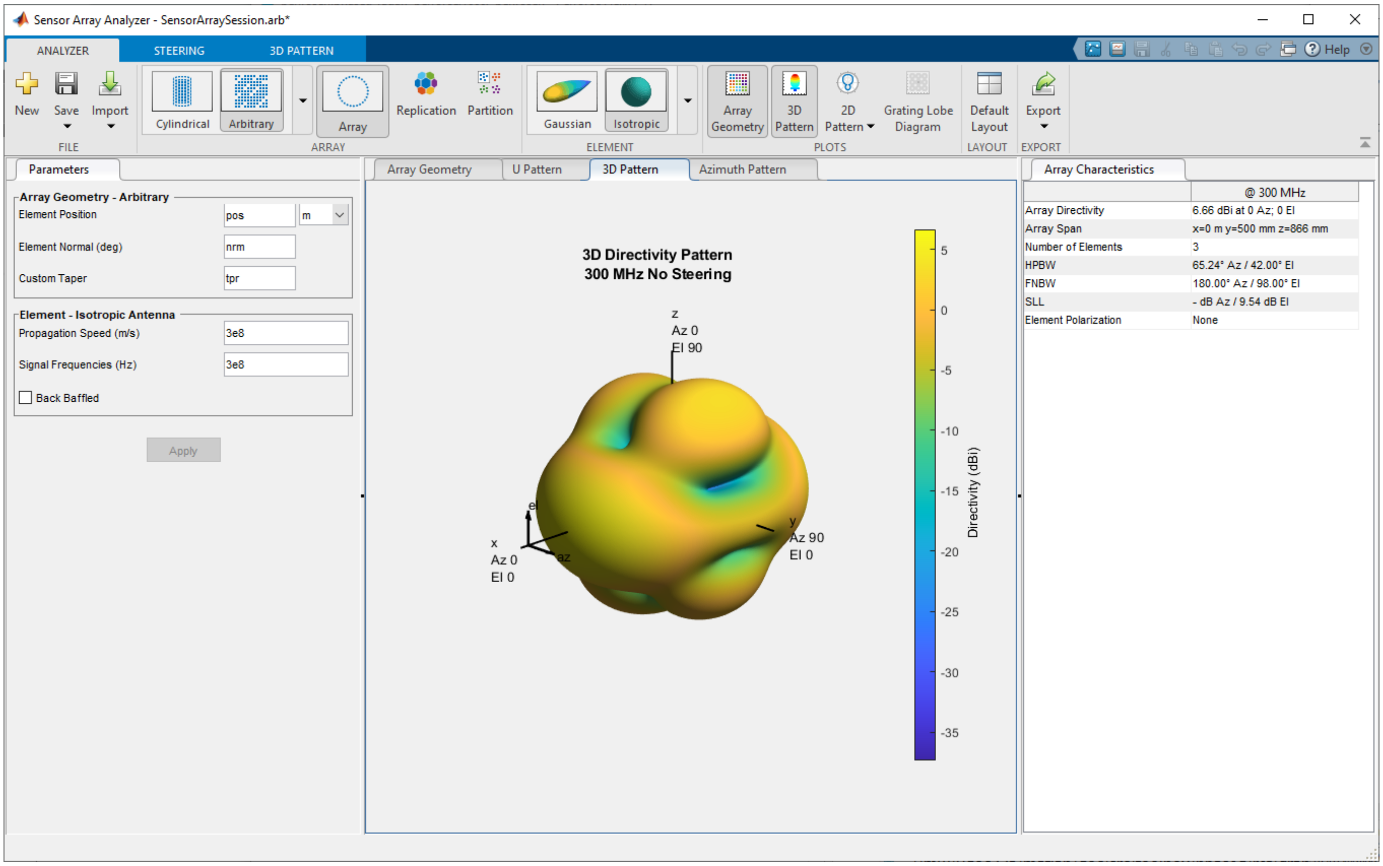
Specify Arbitrary Array Geometry Using Variables

This example illustrates an array with arbitrary geometry specified by MATLAB variables set at the command line. Enter the variables in the appropriate `sensorArrayAnalyzer` fields.

At the MATLAB command line, create an element position array, `pos`, an element normal array, `nrm`, and a taper value array, `tpr`.

```
pos = [0,0,0;0,0.5,0;0,0.5,0.866]
nrm = [0 0 0; 20 20 20];
tpr = [1 1 1];
```

Enter these variables in the appropriate `sensorArrayAnalyzer` fields, click **Apply** button. To show the 3-D array directivity, click **3D Pattern** from the **Plots** tab.



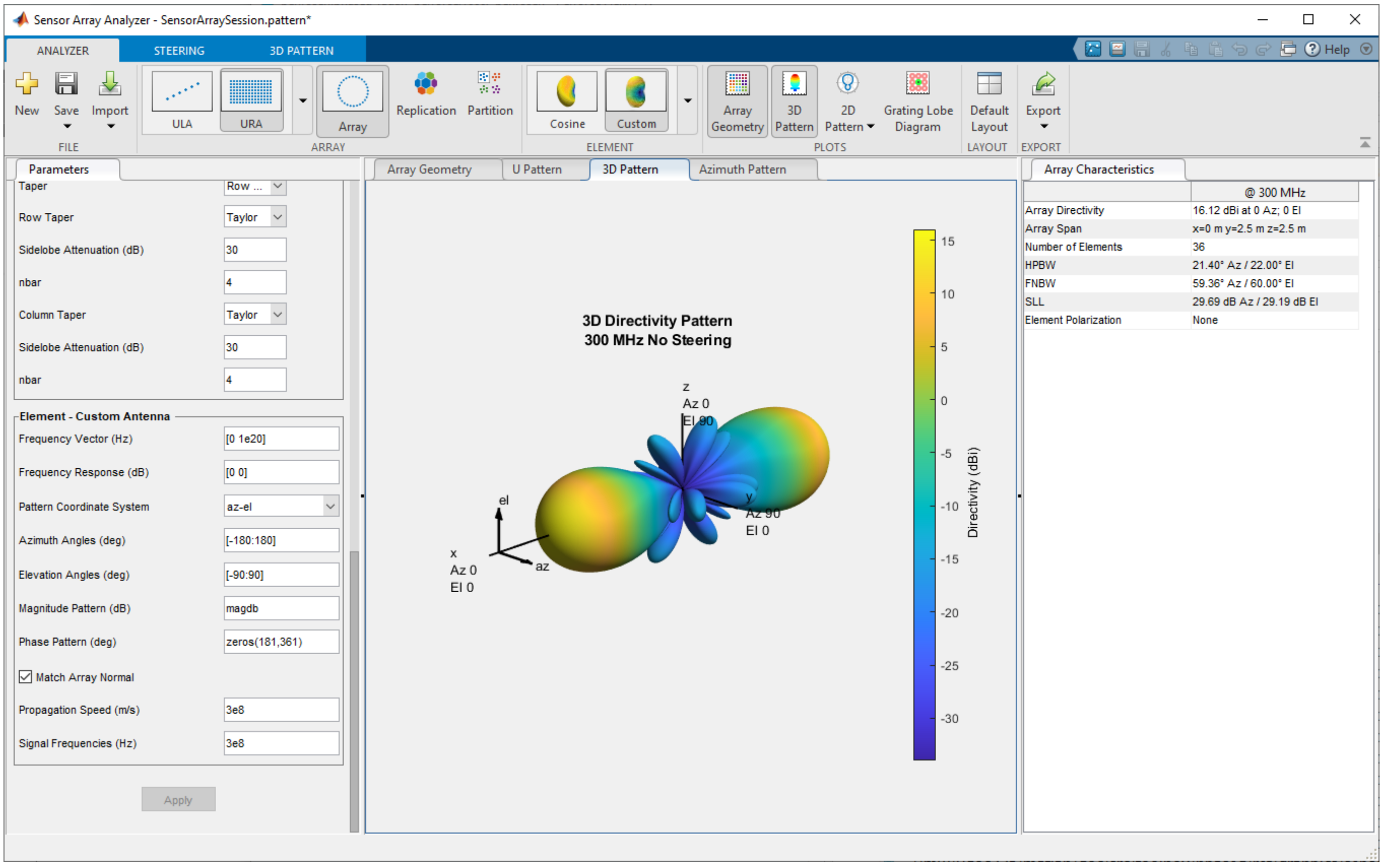
URA With Custom Antenna Element

Use the same parameters as in the "Uniform Rectangular Array (URA)" on page 4-29 example and click the **Apply** button. In the **Element** section of the toolstrip, select **Custom** in the **Antenna** section.

For a custom antenna element, specify the magnitude and phase patterns. Because patterns usually require large matrices, it is better to use the command line to specify the magnitude and phase patterns. The magnitude pattern specified here has directionality along the ±*x*-axes and is a function of azimuth and elevation. The phase pattern is all zeros. Alternatively, you can specify a pattern in terms of phi and theta angles by setting the **Pattern Coordinate System** parameter to `phi-theta`.

```
azpat = cosd([0:360]).^2 + 1;
elpat = cosd([-90:90]') + 1;
mag = elpat*azpat;
magdb = 10*log10(mag);
```

To show the 3-D array directivity, select **3D Pattern** from the **Plots** tab.



- "Array Geometries and Analysis"

Version History

Introduced in R2014b

See Also

Objects

`phased.ReplicatedSubarray` | `phased.PartitionedArray`

Apps

Pulse Waveform Analyzer

Topics

"Array Geometries and Analysis"

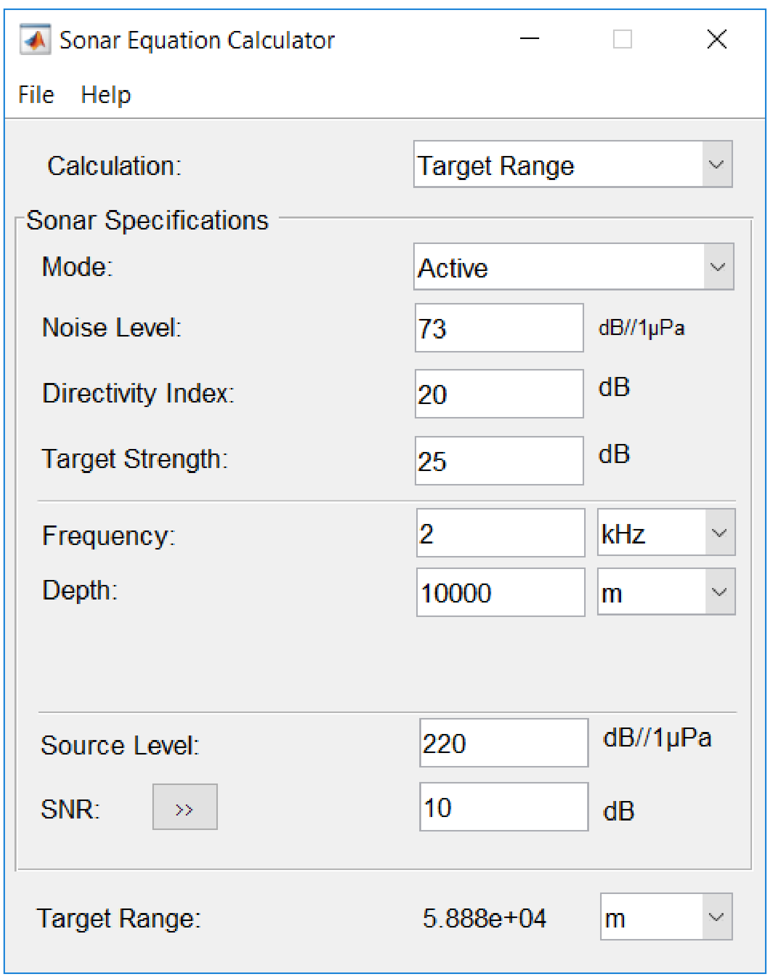
Sonar Equation Calculator

Estimate maximum range, SNR, transmission loss and source level of a sonar system

Description

The **Sonar Equation Calculator** app solves the basic sonar equation for monostatic sonar systems. The sonar equation relates transmission loss (or target range), source level, directivity, noise level, target strength, and signal SNR. You can solve for one of these quantities in terms of the others. Using this app, you can:

- Calculate the received SNR value from transmission loss (or equivalently, target range), source level, and noise level.
- Solve for transmission loss from sonar source level of the sonar, specified received SNR, and array directivity.
- Solve for target range from sonar source level of the sonar, specified received SNR, and array directivity.
- Calculate required source level from target range, source level, and received SNR.



Open the Sonar Equation Calculator App

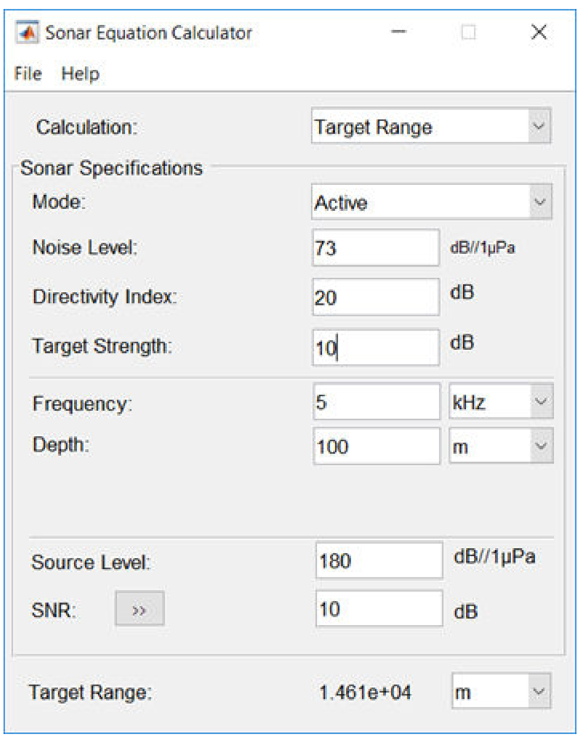
- MATLAB Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon.
- MATLAB command line: Enter `sonarEquationCalculator`.

Examples

Maximum Detection Range of Active Sonar

Compute the maximum detection range of an active monostatic sonar designed to achieve an SNR of at least 10 dB. The operating frequency is 5 kHz, and the source level is 180 dB. Assume that the noise level is 73 dB, the receiver directivity is 20 dB, and the target strength is 10 dB.

- Set **Calculation** to `Target Range`.
- Set **Mode** to `Active`.
- Set **Noise Level** to `73` dB//1µPa.
- Set receiver **Directivity index** to `20` dB.
- Set **Target Strength** to `10` dB.
- Set **Frequency** to `5` kHz.
- Set channel **Depth** to `100` m.
- Set **Source Level** to `180` dB//1µPa.
- Set required **SNR** to `10` dB.



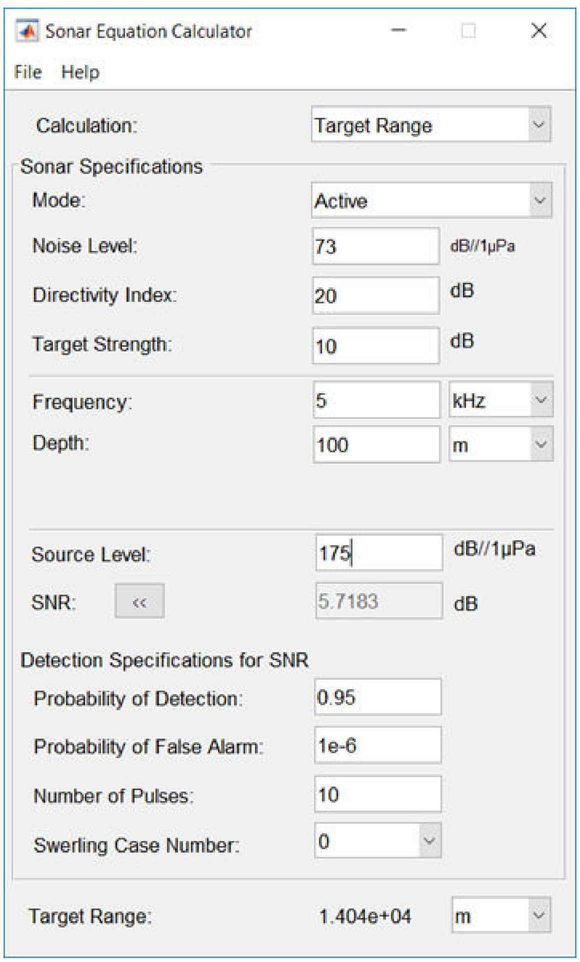
The maximum target range is 14.61 km.

Maximum Detection Range for Multiple Pulses

Use multiple pulses to reduce the source level while maintaining the same maximum target range.

Start with the values set in the “Maximum Detection Range of Active Sonar” on page 4-40 example.

- Click the arrows to the right of the **SNR** label to access the **Detection Specifications for SNR** options.
- Set **Probability of Detection** to `0.95`.
- Set **Probability of False Alarm** to `1e-6`.
- Set **Number of Pulses** to `10`.
- Reduce **Source Level** to `175`.
- Set the **Swerling Case Number** to `0` assuming a nonfluctuating target.

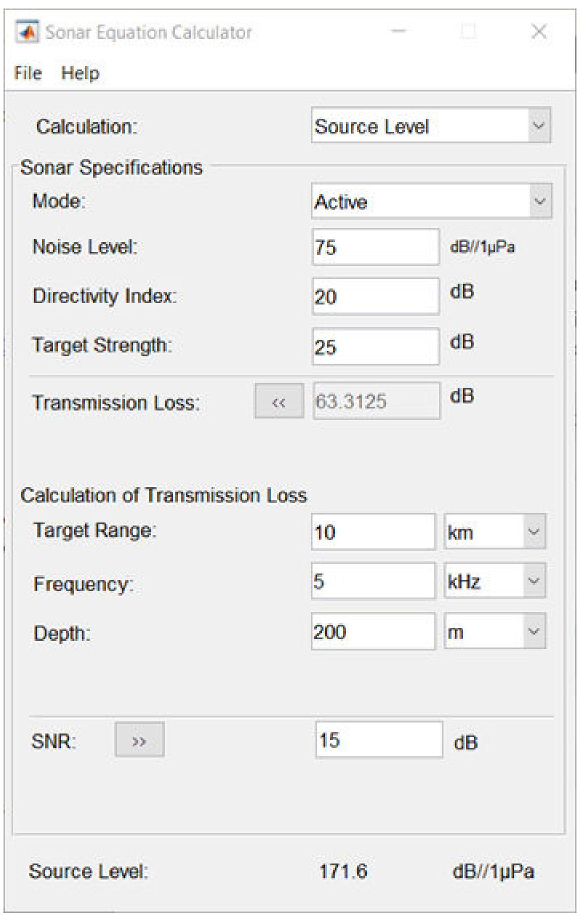


The maximum detection range is 14.81 km, approximately the same as in the previous example, but the source level is reduced by 5 dB.

Required Source Level for Monostatic Sonar

Compute the source level for an active monostatic sonar with a received SNR of 15 dB. The target range is 5 km and the target strength is 25 dB. Assume a 5-kHz sonar frequency.

- Set **Calculation** to `Source Level`.
- Set **Mode** to `Active`.
- Set **Noise Level** to `75`.
- Set receiver **Directivity index** to `20` dB.
- Set **Target Strength** to `25`.
- Click the arrows to the right of the **Transmission Loss** label to access the **Calculation of Transmission Loss** options.
- Set the **Range** to `10.0` km.
- Set the **Frequency** to `5` kHz.
- Set the **Depth** to `200` m.
- Set **SNR** to `15` dB.

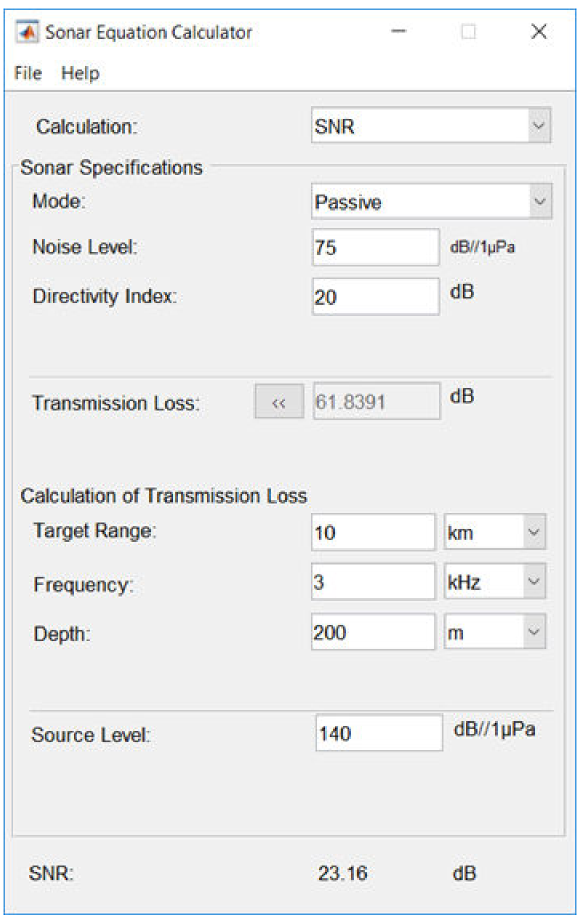


The required source level is 171.6 dB//1 µPa.

Received SNR for Monostatic Sonar

Compute the received SNR for a passive sonar with a source level of 140 db//1 µPa for a source 10.0 km away. Assume a 3-kHz sonar frequency.

- Set **Calculation** to SNR.
- Set **Mode** to `Passive`.
- Set **Noise Level** to 75.
- Set receiver **Directivity index** to `20` dB.
- Click the arrows to the right of the **Transmission Loss** label to access the **Calculation of Transmission Loss** options.
- Set **Range** to `10.0` km.
- Set **Frequency** to 3 kHz.
- Set **Depth** to `200` m.
- Set **Source Level** to `140` dB//1 µPa.

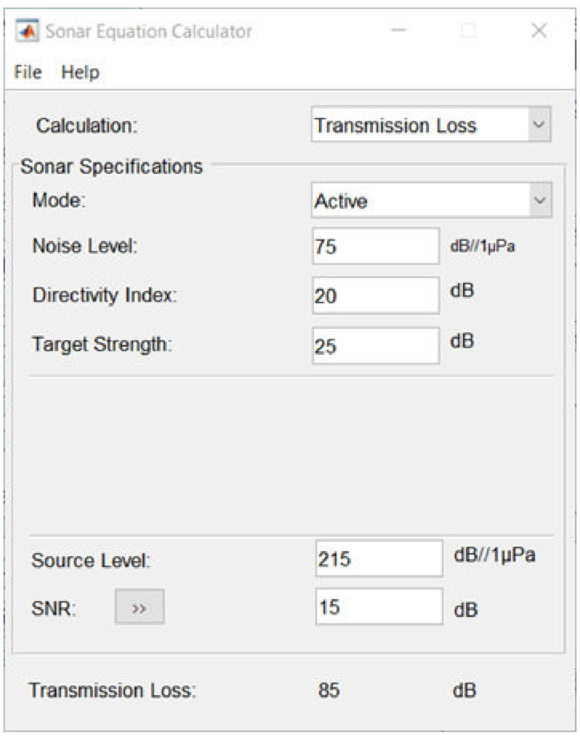


The received SNR is 23.16 dB.

Transmission Loss of Monostatic Active Sonar

Compute the transmission loss for an active sonar that results in an SNR of 15 dB. The source level is 215 dB//1 µPa. Assume that the noise level is 75 dB//1 µPa, the receiver directivity is 20 dB, and the target strength is 10 dB.

- Set **Calculation** to `Transmission Loss`.
- Set **Mode** to `Active`.
- Set **Noise Level** to `75` dB.
- Set receiver **Directivity index** to `20` dB.
- Set **Target Strength** to `25` dB.
- Set **Source Level** to `215` dB.
- Set required **SNR** to `15` dB.



The transmission loss is 85 dB.

Parameters

`Calculation` — Select type of calculation
`Target Range` (default) | `Transmission Loss` | `Source Level` | `SNR`

Select the type of calculation:

- `Target Range` -— solves for the maximum target range based on source level of the sonar and required received SNR.

- `Transmission Loss` -— computes the required transmit power from known target range and required received SNR.
- `Source Level` -— computes the source level from the range or transmission loss, and received SNR.
- `SNR` -— calculates the received SNR value based on known range and transmit power.

`Mode` — Type of sonar
`Active` (default) | `Passive`

Specify whether the sonar is operating in active mode or passive mode. Active mode means that a signal is transmitted from a source, reflects off a target, and returns to the receiver which is collocated with the source. Active mode requires the specification of the reflector target strength. Passive mode means that the signal is transmitted from a source to a receiver along a direct path.

`Noise Level` — Noise Level
`70` (default) | scalar

Noise level at sonar receiver, specified as a scalar. Units are dB//1μP.

`Directivity Index` — Directivity index of receive array or element
`20` (default) | scalar

Directivity index of receive array or element, specified as a scalar. Units are dB.

`Target Strength` — Target strength of reflector
`25` (default) | scalar

Target strength of reflector, specified as a scalar. Units are dB//1m^2.

Dependencies

To enable this parameter, set the **Mode** parameter to `Active`.

`Frequency` — Sound frequency
`2` (default) | positive scalar

Sound frequency, specified as a positive scalar. Default units are in kHz. You can also select `Hz`, `kHz`, or `MHz`.

`Depth` — Water channel depth
`10000` (default) | positive scalar

Water channel depth. Default units are meters. You can select units in `m`, `km`, `mi`, or `nmi`.

`Source Level` — Source level of sonar transmitter
`220` (default) | scalar

Source level of sonar transmitter, specified as a scalar. Units are dB//1 μP.

`SNR` — Output signal-to-noise ratio at receiver
`10` (default)

Specify an SNR value, or calculate an SNR value using the **Detection Specifications for SNR** options. You can calculate the SNR required to achieve a particular probability of detection and probability of false alarm using the Shnidman equation. To calculate the SNR value:

1 Click the arrows to the right of the **SNR** label to access the **Detection Specifications for SNR** options.
2 Enter values for **Probability of Detection**, **Probability of False Alarm**, **Number of Pulses**, and **Swerling Case Number**.

Dependencies

To enable this parameter, set **Calculation** to `Target Range`, `Transmission Loss`, or `Source Level`.

`Probability of Detection` — Detection probability used to estimate SNR
`0.81029` (default)

Specify the detection probability used to estimate SNR using the Shnidman equation.

Dependencies

To enable this parameter, set **Calculation** to `Target Range`, `Transmission Loss`, or `Source Level`, and select the **Detection Specifications for SNR** options for the **SNR** parameter.

`Probability of False Alarm` — False alarm probability used to estimate SNR
`0.001` (default)

Specify the false alarm probability used to estimate SNR using the Shnidman equation.

Dependencies

To enable this parameter, set **Calculation** to `Target Range`, `Transmission Loss`, or `Source Level`, and access the **Detection Specifications for SNR** options for the **SNR** parameter.

`Number of Pulses` — Number of pulses used to estimate SNR
`1` (default)

Specify the number of pulses. You can specify multiple pulses for noncoherent integration in the Shnidman equation.

Dependencies

To enable this parameter, set **Calculation** to `Target Range`, `Transmission Loss`, or `Source Level`, and select the **Detection Specifications for SNR** options for the **SNR** parameter.

`Swerling Case Number` — Swerling case number used estimate SNR
`0` (default) | `1` | `2` | `3` | `4`

Specify the Swerling case number used to estimate SNR using the Shnidman equation. Swerling numbers characterize the detection problem for fluctuating pulses in terms of:

- a decorrelation model for the received pulses.
- the distribution of scatterers affecting the probability density function (pdf) of the target radar cross section (RCS).

The Swerling cases include two decorrelation models (scan-to-scan or pulse-to-pulse) and two radar cross section pdfs (based on the presence or absence of a dominant scatterer):

- **`0`** – Nonfluctuating pulses.

- **1** – Scan-to-scan decorrelation: Several randomly distributed scatterers with no dominant scatterer described by a Rayleigh/exponential PDF.
- **2** – Pulse-to-pulse decorrelation: Several randomly distributed scatterers with no dominant scatterer described by a Rayleigh/exponential PDF.
- **3** – Scan-to-scan decorrelation: Several scatterers with one dominant scatterer described by a chi-square PDF with 4 degrees of freedom.
- **4** – Pulse-to-pulse decorrelation: Several scatterers with one dominant scatterer described by a chi-square PDF with 4 degrees of freedom.

Dependencies

To enable this parameter, set **Calculation** to `Target Range`, `Transmission Loss`, or `Source Level`, and select the **Detection Specifications for SNR** options for the **SNR** parameter.

`Transmission Loss` — Transmission loss in channel
`78.0614` (default) | scalar

Transmission loss in channel, specified as a scalar. Units are dB. For passive sonar modeling, specify a one-way transmission loss. For active sonar modeling, specify a two-way transmission loss. You can specify a transmission loss value, or calculate transmission loss using the **Calculation of Transmission Loss** options.

To calculate transmission loss:

1. Click the arrows to the right of the **Transmission Loss** label to access the **Calculation of Transmission Loss** menu.
2. Enter values for **Target Range**, **Frequency**, and **Depth**.

Dependencies

To enable this parameter, set **Calculation** to `Source Level` or `SNR`.

`Target Range` — Target range
`10000` (default) | positive scalar

Target range, specified as a positive scalar. When **Mode** is `Passive`, target range is from source to receiver. When **Mode** is `Active`, target range is from source to reflecting target. Default units are in meters. You can also select `km`, `mi`, or `nmi`.

Dependencies

To enable this parameter, set the **Calculation** parameter to `Source Level` or `SNR` and click the arrow next to **Calculation of Transmission Loss**.

`Frequency` — Signal frequency
`2` (default) | positive scalar

Signal frequency, specified as a positive scalar. Default units are in kHz. You can also select `Hz`, `kHz`, and `MHz`.

Dependencies

To enable this parameter, set **Calculation** to `Source Level` or `SNR` and click the arrow next to **Calculation of Transmission Loss**.

`Depth` — Channel depth
`10000` (default) | positive scalar

Channel depth, specified as a positive scalar. Default units are in meters. You can also select `km`, `mi`, and `nmi`.

Dependencies

To enable this parameter, set the **Calculation** parameter to `Source Level` or `SNR` and click the arrow next to **Calculation of Transmission Loss**.

Version History

Introduced in R2017b

See Also

Apps
Sensor Array Analyzer

Functions
`range2tl` | `sonareqsl` | `sonareqsnr` | `sonareqtl` | `tl2range`

Topics
"Sonar Equation"