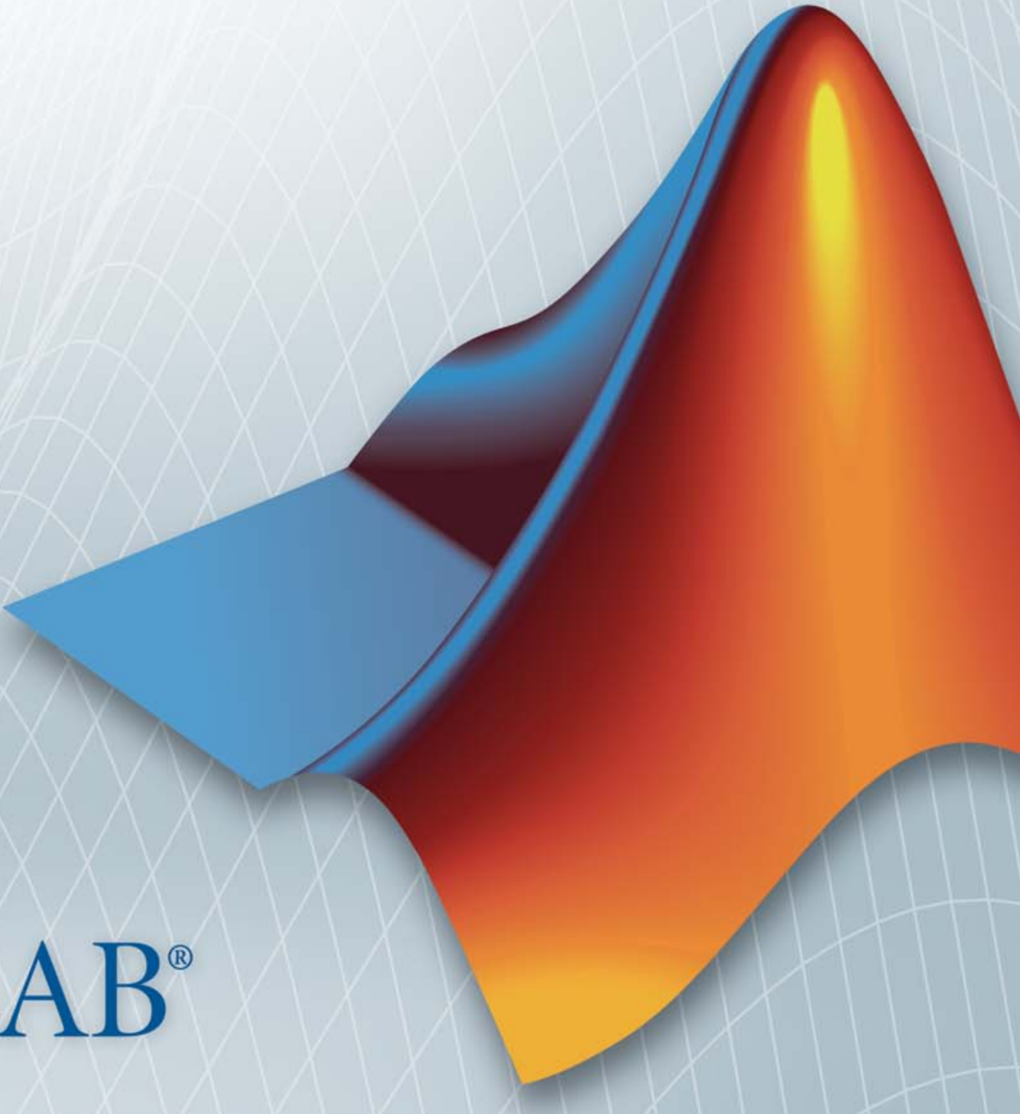
Phased Array System Toolbox™

Reference

R2012b



MATLAB®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

For contact information about worldwide offices, see the MathWorks Web site.

Phased Array System Toolbox™ Reference

© COPYRIGHT 2011–2012 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See `www.mathworks.com/trademarks` for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

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)

Contents

1 Function Reference

2 System Object Reference

1

Function Reference

Array Analysis (p. 1-2)	Analyze array response
Array Antenna Elements (p. 1-3)	Model antenna elements
Coordinate Systems and Motion Modeling (p. 1-4)	Motion managers
Detection (p. 1-5)	Signal detection and matched filtering
Environment Models (p. 1-6)	Modeling signal propagation
Radar Analysis (p. 1-7)	Radar equation modeling
Receiver Models (p. 1-8)	Model a phased array receiver
Space-Time Adaptive Processing (p. 1-9)	Angle-Doppler processing
Transmitter Models (p. 1-10)	Model a pulse transmitter
Utilities (p. 1-11)	General utility functions
Waveforms (p. 1-12)	Waveform analysis

Array Analysis

`az2broadside`	Convert azimuth angle to broadside angle
`broadside2az`	Convert broadside angle to azimuth angle

Array Antenna Elements

`aperture2gain`	Convert effective aperture to gain
`azel2phithetapat`	Convert radiation pattern from azimuth/elevation to phi/theta form
`azel2uvpat`	Convert radiation pattern from azimuth/elevation form to u/v form
`gain2aperture`	Convert gain to effective aperture
`phitheta2azelpat`	Convert radiation pattern from phi/theta form to azimuth/elevation form
`phitheta2uvpat`	Convert radiation pattern from phi/theta form to u/v form
`uv2azelpat`	Convert radiation pattern from u/v form to azimuth/elevation form
`uv2phithetapat`	Convert radiation pattern from u/v form to phi/theta form

Coordinate Systems and Motion Modeling

`azel2phitheta`	Convert angles from azimuth/elevation form to phi/theta form
`azel2uv`	Convert azimuth/elevation angles to u/v coordinates
`dop2speed`	Convert Doppler shift to speed
`global2localcoord`	Convert global to local coordinates
`local2globalcoord`	Convert local to global coordinates
`phitheta2azel`	Convert angles from phi/theta form to azimuth/elevation form
`phitheta2uv`	Convert phi/theta angles to u/v coordinates
`radialspeed`	Relative radial speed
`rangeangle`	Range and angle calculation
`speed2dop`	Convert speed to Doppler shift
`uv2azel`	Convert u/v coordinates to azimuth/elevation angles
`uv2phitheta`	Convert u/v coordinates to phi/theta angles

Detection

`albersheim`	Required SNR using Albersheim's equation
`npwgnthresh`	Detection SNR threshold for signal in white Gaussian noise
`pulsint`	Pulse integration
`rocpfa`	Receiver operating characteristic curves by false-alarm probability
`rocsnr`	Receiver operating characteristic curves by SNR
`shnidman`	Required SNR using Shnidman's equation
`stretchfreq2rng`	Convert frequency offset to range

Environment Models

`billingsleyicm`	Billingsley's intrinsic clutter motion (ICM) model
`depressionang`	Depression angle of surface target
`effearthradius`	Effective earth radius
`fspl`	Free space path loss
`grazingang`	Grazing angle of surface target
`horizonrange`	Horizon range
`surfacegamma`	Gamma value for different terrains
`surfclutterrcs`	Surface clutter radar cross section (RCS)

Radar Analysis

`radareqpow`	Peak power estimate from radar equation
`radareqrng`	Maximum theoretical range estimate
`radareqsnr`	SNR estimate from radar equation

Receiver Models

`noisepow`	Receiver noise power
`systemp`	Receiver system-noise temperature

Space-Time Adaptive Processing

`dopsteeringvec`	Doppler steering vector

Transmitter Models

Utilities

`delayseq`	Delay or advance sequence
`physconst`	Physical constants
`unigrid`	Uniform grid
`val2ind`	Uniform grid index

Waveforms

`ambgfun`	Ambiguity function

2

System Object Reference

Array Analysis (p. 2-2)	Analyze array response
Array Antenna Elements (p. 2-3)	Model antenna elements
Array Microphone Elements (p. 2-4)	Model microphone elements
Array Design (p. 2-5)	Design array geometries
Beamformers (p. 2-6)	Beamforming
Collector (p. 2-7)	Model incident waveforms at arrays
Coordinate Systems and Motion Modeling (p. 2-8)	Motion managers
Detection (p. 2-9)	Signal detection and matched filtering
Direction of Arrival (DOA) (p. 2-10)	DOA estimation
Environment Models (p. 2-11)	Model propagation environments
Jammer Models (p. 2-12)	Model signal jammers
Radiator (p. 2-13)	Model signal radiation
Receiver Models (p. 2-14)	Model a phased array receiver
Space-Time Adaptive Processing (p. 2-15)	Implement space-time adaptive processing
Target Models (p. 2-16)	Model targets
Transmitter Models (p. 2-17)	Model a pulse transmitter
Waveforms (p. 2-18)	Construct pulse waveforms
Define New System Objects (p. 2-19)	Create new kinds of System objects

Array Analysis

phased.ArrayGain	Sensor array gain
phased.ArrayResponse	Sensor array response
phased.ElementDelay	Sensor array element delay estimator
phased.SteeringVector	Sensor array steering vector

Array Antenna Elements

phased.CosineAntennaElement	Cosine antenna
phased.CustomAntennaElement	Custom antenna
phased.IsotropicAntennaElement	Isotropic antenna

Array Microphone Elements

phased.CustomMicrophoneElement	Custom microphone
phased.OmnidirectionalMicrophoneElement	Omnidirectional microphone

Array Design

phased.ConformalArray	Conformal array
phased.PartitionedArray	Phased array partitioned into subarrays
phased.ReplicatedSubarray	Phased array formed by replicated subarrays
phased.ULA	Uniform linear array
phased.URA	Uniform rectangular array

Beamformers

phased.FrostBeamformer	Frost beamformer
phased.LCMVBeamformer	Narrowband LCMV beamformer
phased.MVDRBeamformer	Narrowband MVDR (Capon) beamformer
phased.PhaseShiftBeamformer	Narrowband phase shift beamformer
phased.SubbandPhaseShiftBeamformer	Subband phase shift beamformer
phased.TimeDelayBeamformer	Time delay beamformer
phased.TimeDelayLCMVBeamformer	Time delay LCMV beamformer

Collector

phased.Collector	Narrowband signal collector
phased.WidebandCollector	Wideband signal collector

Coordinate Systems and Motion Modeling

phased.Platform	Motion platform

Detection

phased.CFARDetector	Constant false alarm rate (CFAR) detector
phased.MatchedFilter	Matched filter
phased.StretchProcessor	Stretch processor for linear FM waveform
phased.TimeVaryingGain	Time varying gain control

Direction of Arrival (DOA)

phased.BeamscanEstimator	Beamscan spatial spectrum estimator for ULA
phased.BeamscanEstimator2D	2-D beamscan spatial spectrum estimator
phased.BeamspaceESPRITEstimator	Beamspace ESPRIT direction of arrival (DOA) estimator
phased.ESPRITEstimator	ESPRIT direction of arrival (DOA) estimator
phased.MVDREstimator	MVDR (Capon) spatial spectrum estimator for ULA
phased.MVDREstimator2D	2-D MVDR (Capon) spatial spectrum estimator
phased.RootMUSICEstimator	Root MUSIC direction of arrival (DOA) estimator
phased.RootWSFEstimator	Root WSF direction of arrival (DOA) estimator
phased.SumDifferenceMonopulseTracker	Sum and difference monopulse for ULA
phased.SumDifferenceMonopulseTracker2D	Sum and difference monopulse for URA

Environment Models

phased.ConstantGammaClutter	Constant gamma clutter simulation
phased.FreeSpace	Free space environment
phased.gpu.ConstantGammaClutter	Constant gamma clutter simulation on GPU

Jammer Models

phased.BarrageJammer	Barrage jammer

Radiator

phased.Radiator	Narrowband signal radiator

Receiver Models

phased.ReceiverPreamp	Receiver preamp

Space-Time Adaptive Processing

phased.ADPCACanceller	Adaptive DPCA (ADPCA) pulse canceller
phased.AngleDopplerResponse	Angle-Doppler response
phased.DPCACanceller	Displaced phase center array (DPCA) pulse canceller
phased.STAPSMIBeamformer	Sample matrix inversion (SMI) beamformer

Target Models

phased.RadarTarget	Radar target

Transmitter Models

phased.Transmitter Transmitter

Waveforms

phased.FMCWWaveform	FMCW Waveform
phased.LinearFMWaveform	Linear FM pulse waveform
phased.PhaseCodedWaveform	Phase-coded pulse waveform
phased.RectangularWaveform	Rectangular pulse waveform
phased.SteppedFMWaveform	Stepped FM pulse waveform

Define New System Objects

getDiscreteStateImpl	Discrete state property values
getNumInputsImpl	Number of input arguments passed to step and setup methods
getNumOutputsImpl	Number of outputs returned by method
isDoneImpl	End-of-data flag
isInactivePropertyImpl	Active or inactive flag for properties
loadObjectImpl	Load saved System object from MAT file
matlab.System	Base class for System objects
matlab.system.mixin.FiniteSource	Finite source mixin class
matlab.system.StringSet	Set of valid string values
processTunedPropertiesImpl	Action when tunable properties change
releaseImpl	Release resources
resetImpl	Reset System object™ states
saveObjectImpl	Save System object in MAT file
setProperties	Set property values from name-value pair inputs
setupImpl	Initialize System object
stepImpl	System output and state update equations
validateInputsImpl	Validate inputs to step method
validatePropertiesImpl	Validate property values

3

Alphabetical List

matlab.System

Purpose

Base class for System objects

Description

`matlab.System` is the base class for System objects. In your class definition file, you must subclass your object from this base class (or from another class that derives from this base class). Subclassing allows you to use the implementation and service methods provided by this base class to build your object. You use this syntax as the first line of your class definition file to directly inherit from the `matlab.System` base class, where `ObjectName` is the name of your object:

```
classdef ObjectName < matlab.System
```

Note You must set `Access=protected` for each `matlab.System` method you use in your code.

Methods

getDiscreteStateImpl	Discrete state property values
getNumInputsImpl	Number of input arguments passed to step and setup methods
getNumOutputsImpl	Number of outputs returned by method
isInactivePropertyImpl	Active or inactive flag for properties
loadObjectImpl	Load saved System object from MAT file
processTunedPropertiesImpl	Action when tunable properties change
releaseImpl	Release resources
resetImpl	Reset System object states
saveObjectImpl	Save System object in MAT file

setProperties	Set property values from name-value pair inputs
setupImpl	Initialize System object
stepImpl	System output and state update equations
validateInputsImpl	Validate inputs to step method
validatePropertiesImpl	Validate property values

Attributes

In addition to the attributes available for MATLAB® objects, you can apply the following attributes to any property of a custom System object.

`Nontunable`	After an object is locked (after `step` or `setup` has been called), use `Nontunable` to prevent a user from changing that property value. By default, all properties are tunable. The `Nontunable` attribute is useful to lock a property that has side effects when changed. This attribute is also useful for locking a property value assumed to be constant during processing. You should always specifiy properties that affect the number of input or output ports as `Nontunable`.
`Logical`	Use `Logical` to limit the property value to a logical, scalar value. Any scalar value that can be converted to a logical is also valid, such as 0 or 1.
`PositiveInteger`	Use `PositiveInteger` to limit the property value to a positive integer value.
`DiscreteState`	Use `DiscreteState` to mark a property so it will display its state value when you use the `getDiscreteState` method.

To learn more about attributes, see “Property Attributes” in the MATLAB Object-Oriented Programming documentation.

Examples

Create a simple System object, `AddOne`, which subclasses from `matlab.System`. You place this code into a MATLAB file, `AddOne.m`.

```
classdef AddOne < matlab.System
%ADDONE Compute an output value that increments the input by one

    methods (Access=protected)
       % stepImpl method is called by the step method.
       function y = stepImpl(~,x)
          y = x + 1;
       end
    end
end
```

To use this object, create an instance of `AddOne`, provide an input, and use the `step` method:

```
hAdder = AddOne;
x = 1;
y = step(hAdder,x)
```

Assign the `Nontunable` attribute to the `InitialValue` property, which you define in your class definition file.

```
properties (Nontunable)
   InitialValue
end
```

```
| | matlab.system.StringSet | | | |
matlab.system.mixin.FiniteSource
```

How To

- “Object-Oriented Programming”

- Class Attributes
- Property Attributes
- “Method Attributes”
-
-
-
- “Define Basic System Objects”
-
-
-
- “Define Property Attributes”

matlab.System.getDiscreteStateImpl

Purpose

Discrete state property values

Syntax

`s = getDiscreteStateImpl(obj)`

Description

`s = getDiscreteStateImpl(obj)` returns a struct `s` of state values. The field names of the struct are the object's `DiscreteState` property names. To restrict or change the values returned by `getDiscreteState` method, you can override this `getDiscreteStateImpl` method. End users cannot specify scaled double fi objects as inputs to discrete state properties.

`getDiscreteStatesImpl` is called by the `getDiscreteState` method, which is called by the `setup` method.

Note You must set `Access=protected` for this method.

Input Arguments

obj

System object handle

Output Arguments

s

Struct of state values.

Examples

```
methods (Access=protected)
   function s = getDiscreteState(obj)
   end
end
```

`| | setupImpl`

-
-
- "Define Property Attributes"

Purpose

Number of input arguments passed to step and setup methods

Syntax

```
num = getNumInputsImpl(obj)
```

Description

`num = getNumInputsImpl(obj)` returns the number of inputs `num` (excluding the System object handle) expected by the `step` method. The default implementation returns 1, which requires one input from the user, in addition to the System object handle. To specify a value other than 1, you must use include the `getNumInputsImpl` method in your class definition file.

`getNumInputsImpl` is called by the `getNumInputs` method and by the `setup` method if the number of inputs has not been determined already.

Note You must set `Access=protected` for this method.

Input Arguments

obj

System object handle

Output Arguments

num

Number of inputs expected by the `step` method for the specified object.

Default: 1

Examples

Specify the number of inputs (2, in this case) expected by the `step` method.

```
methods (Access=protected)
   function num = getNumInputsImpl(obj)
      num = 2;
   end
end
```

Specify that the `step` method will not accept any inputs.

```
methods (Access=protected)
   function num = getNumInputsImpl(~)
      num = 0;
   end
end
```

`| | setupImpl | | | | stepImpl | | | | getNumOutputsImpl`

-
-
- “Change Number of Step Inputs or Outputs”

Purpose

Number of outputs returned by `step` method

Syntax

`num = getNumOutputsImpl (obj)`

Description

`num = getNumOutputsImpl (obj)` returns the number of outputs from the `step` method. The default implementation returns 1 output. To specify a value other than 1, you must use include the `getNumOutputsImpl` method in your class definition file.

`getNumOutputsImpl` is called by the `getNumOutputs` method, if the number of outputs has not been determined already.

Note You must set `Access=protected` for this method.

Input Arguments

obj

System object handle

Output Arguments

num

Number of outputs to be returned by the `step` method for the specified object.

Examples

Specify the number of outputs (2, in this case) returned from the `step` method.

```
methods (Access=protected)
   function num = getNumOutputsImpl(obj)
      num = 2;
   end
end
```

Specify that the `step` method does not return any outputs.

```
methods (Access=protected)
```

```
   function num = getNumOutputsImpl(~)
      num = 0;
   end
end
```

| | `stepImpl` | | | | `getNumInputsImpl` | | | | `setupImpl`

-
-
- "Change Number of Step Inputs or Outputs"

matlab.System.isInactivePropertyImpl

Purpose

Active or inactive flag for properties

Syntax

`flag = isInactivePropertyImpl(obj,prop)`

Description

`flag = isInactivePropertyImpl(obj,prop)` specifies whether a property is inactive for the current object configuration. An *inactive property* is a property that is not relevant to the object, given the values of other properties. Inactive properties are not shown if you use the `disp` method to display object properties. If you attempt to use public access to directly access or use `get` or `set` on an inactive property, a warning occurs.

`isInactiveProperty` is called by the `disp` method and by the `get` and `set` methods.

Note You must set `Access=protected` for this method.

Input Arguments

obj

System object handle

prop

Property name

Output Arguments

flag

Logical scalar value indicating whether the input property `prop` is inactive for the current object configuration.

Examples

Display the `InitialValue` property only when the `UseRandomInitialValue` property value is `false`.

```
methods (Access=protected)
  function flag = isInactivePropertyImpl(obj,propertyName)
    if strcmp(propertyName,'InitialValue')
      flag = obj.UseRandomInitialValue;
```

```
    else
      flag = false;
    end
  end
end
```

| | `setProperties`

-
-
- “Hide Inactive Properties”

Purpose

Load saved System object from MAT file

Syntax

`loadObjectImpl(obj)`

Description

`loadObjectImpl(obj)` loads a saved System object, `obj`, from a MAT file. Your `loadObjectImpl` method should correspond to your `saveObjectImpl` method to ensure that all saved properties and data are loaded.

Input Arguments

obj

System object handle

Examples

Load a saved System object. In this case, the object contains a child object, protected and private properties, and a discrete state.

```
methods(Access=protected)
   function loadObjectImpl(obj, s, wasLocked)
      % Load child System objects
      obj.child = matlab.System.loadObject(s.child);

      % Save protected & private properties
      obj.protected = s.protected;
      obj.pdependentprop = s.pdependentprop;

      % Save state only if locked when saved
      if wasLocked
        obj.state = s.state;
      end

      % Call base class method
      loadObjectImpl@matlab.System(obj,s,wasLocked);
    end
  end
```

How To

- “Load System Object”

- “Save System Object”

matlab.System.processTunedPropertiesImpl

Purpose

Action when tunable properties change

Syntax

`processTunedPropertiesImpl(obj)`

Description

`processTunedPropertiesImpl(obj)` specifies the actions to perform when one or more tunable property values change. This method is called as part of the next call to the `step` method after a tunable property value changes. A property is tunable only if its `Nontunable` attribute is `false`, which is the default.

`processTunedPropertiesImpl` is called by the `step` method.

Note You must set `Access=protected` for this method.

Tips

Use this method when a tunable property affects a different property value. For example, two property values determine when to calculate a lookup table. You want to perform that calculation when either property changes. You also want the calculation to be done only once if both properties change before the next call to the `step` method.

Input Arguments

obj

System object handle

Examples

Use `processTunedPropertiesImpl` to recalculate the lookup table if the value of either the `NumNotes` or `MiddleC` property changes.

```
methods (Access=protected)
   function processTunedPropertiesImpl(obj)
      % Generate a lookup table of note frequencies
      obj.pLookupTable = obj.MiddleC * (1+log(1:obj.NumNotes)/log(12)
    end
end
```

`| | validatePropertiesImpl | | | | setProperties`

-
-
- “Validate Property and Input Values”
-
-
-
- “Define Property Attributes”

matlab.System.releaseImpl

Purpose

Release resources

Syntax

`releaseImpl(obj)`

Description

`releaseImpl(obj)` releases any resources used by the System object, such as file handles. This method also performs any necessary cleanup tasks. To release resources for a System object, you must use `releaseImpl` instead of a destructor.

`releaseImpl` is called by the `release` method. `releaseImpl` is also called when the object is deleted or cleared from memory, or when all references to the object have gone out of scope.

Note You must set `Access=protected` for this method.

Input Arguments

obj

System object handle

Examples

Use the `releaseImpl` method to close a file.

```
methods (Access=protected)
   function releaseImpl(obj)
     fclose(obj.pFileID);
   end
end
```

`| | resetImpl`

-
-
- “Release System Object Resources”

matlab.System.resetImpl

Purpose

Reset System object states

Syntax

`resetImpl(obj)`

Description

`resetImpl(obj)` defines the state reset equations for the System object. Typically you reset the states to a set of initial values.

`resetImpl` is called by the `reset` method. It is also called by the `setup` method, after the `setupImpl` method.

Note You must set `Access=protected` for this method.

Input Arguments

obj

System object handle

Examples

Use the `reset` method to reset the counter `pCount` property to zero.

```
methods (Access=protected)
   function resetImpl(obj)
      obj.pCount = 0;
   end
end
```

| | releaseImpl

-
-
- "Reset Algorithm State"

matlab.System.saveObjectImpl

Purpose

Save System object in MAT file

Syntax

`saveObjectImpl(obj)`

Description

`saveObjectImpl(obj)` defines what System object `obj` property and state values are saved in a MAT file when a user calls `save` on that object. `save` calls `saveObject`, which then calls `saveObjectImpl`. If you do not define a `saveObjectImpl` method for your System object class, only public properties are saved. To save any private or protected properties or state information, you must define a `saveObjectImpl` in your class definition file.

You should save the state of an object only if the object is locked. When the user loads that saved object, it loads in that locked state.

To save child object information, you use the associated `saveObject` method within the `saveObjectImpl` method.

End users can use `load`, which calls `loadObjectImpl` to load a System object into their workspace.

Input Arguments

obj

System object handle

Examples

Define what is saved for the System object. Call the base class version of `saveObjectImpl` to save public properties. Then, save any child System objects and any protected and private propertes. Finally, save the state, if the object is locked.

```
methods(Access=protected)
   function s = saveObjectImpl(obj)
      s = saveObjectImpl@matlab.System(obj);
      s.child = matlab.System.saveObject(obj.child);
      s.protected = obj.protected;
      s.pdependentprop = obj.pdependentprop;
      if isLocked(obj)
         s.state = obj.state;
```

```
        end
    end
end
```

How To

- “Save System Object”
- “Load System Object”

matlab.System.setProperties

Purpose

Set property values from name-value pair inputs

Syntax

```
setProperties(obj,numargs,name1,value1,name2,value2,...)
setProperties(obj,numargs,arg1,...,argm,name1,value1,name2,
   value2,...)
```

Description

`setProperties(obj,numargs,name1,value1,name2,value2,...)` provides the name-value pair inputs to the System object constructor. Use this syntax if every input must specify both name and value.

Note To allow standard name-value pair handling at construction, define `setProperties` for your System object.

`setProperties(obj,numargs,arg1,...,argm,name1,value1,name2,value2,...` provides the value-only inputs, followed by the name-value pair inputs to the System object during object construction. Use this syntax if you want to allow users to specify one or more inputs by their values only.

Input Arguments

obj

System object handle

numargs

Number of inputs passed in by the object constructor

name*

Name of property

value*

Value of the property

arg*

Value of property (for value-only input to the object constructor)

Examples

Set up the object so users can specify property values via name-value pairs when constructing the object.

```
methods
   function obj = MyFile(varargin)
      setProperties(obj,nargin,varargin{:});
   end
end
```

-
-
- "Set Property Values at Construction Time"

matlab.System.setupImpl

Purpose

Initialize System object

Syntax

`setupImpl(obj,input1, input2,...)`

Description

`setupImpl(obj,input1, input2,...)` sets up a System object. To acquire resources for a System object, you must use `setupImpl` instead of a constructor. `setupImpl` executes the first time the `step` method is called on an object after that object has been created. It also executes the next time `step` is called after an object has been released. . The number of inputs must match the number of inputs defined in the `getNumInputsImpl` method. You pass the inputs into `setupImpl` to use the input sizes, datatypes, etc. in the one-time calculations.

`setupImpl` is called by the `setup` method, which is done automatically as the first subtask of the `step` method on an unlocked System object.

Note You must set `Access=protected` for this method.

Tips

To validate properties or inputs use the `validatePropertiesImpl`, `validateInputsImpl`, or `setProperties` methods. Do not include validation in `setupImpl`.

Input Arguments

obj

System object handle

input*

Inputs to the `setup` method

Examples

Open a file for writing using the `setupImpl` method.

```
methods (Access=protected)
   function setupImpl(obj,data)
      obj.pFileID = fopen(obj.Filename, 'wb');
      if obj.pFileID < 0
```

```
            error('Opening the file failed');
        end
    end
end
```

| | validatePropertiesImpl | | | | validateInputsImpl | | | | setProperties

-
-
- “Initialize Properties and Setup One-Time Calculations”
-
-
-
- “Set Property Values at Construction Time”

matlab.System.stepImpl

Purpose

System output and state update equations

Syntax

`[output1,output2,...] = stepImpl(obj,input1,input2,...)`

Description

`[output1,output2,...]  = stepImpl(obj,input1,input2,...)` defines the algorithm to execute when you call the `step` method on the specified object `obj`. The `step` method calculates the outputs and updates the object's state values using the inputs, properties, and state update equations.

`stepImpl` is called by the `step` method.

Note You must set `Access=protected` for this method.

Tips

The number of input arguments and output arguments must match the values returned by the `getNumInputsImpl` and `getNumOutputsImpl` methods, respectively

Input Arguments

obj

System object handle

input*

Inputs to the `step` method

Output Arguments

output

Output returned from the `step` method.

Examples

Use the `stepImpl` method to increment two numbers.

```
methods (Access=protected)
   function [y1,y2] = stepImpl(obj,x1,x2)
      y1 = x1 + 1;
      y2 = x2 + 1;
    end
```

`| | getNumInputsImpl | | | | getNumOutputsImpl | | | | validateInputsImpl`

-
-
- "Define Basic System Objects"
-
-
-
- "Change Number of Step Inputs or Outputs"

Purpose

Validate inputs to step method

Syntax

`validateInputsImpl(obj,input1,input2,...)`

Description

`validateInputsImpl(obj,input1,input2,...)` validates inputs to the `step` method at the beginning of initialization Validation includes checking data types, complexity, cross-input validation, and validity of inputs controlled by a property value.

`validateInputsImpl` is called by the `setup` method before `setupImpl`. `validateInputsImpl` executes only once.

Note You must set `Access=protected` for this method.

Input Arguments

obj

System object handle

input*

Inputs to the `setup` method

Examples

Validate that the input is numeric.

```
methods (Access=protected)
   function validateInputsImpl(~,x)
      if ~isnumeric(x)
         error('Input must be numeric');
      end
   end
end
```

| | validatePropertiesImpl | | | | setupImpl

.

.

- “Validate Property and Input Values”

Purpose

Validate property values

Syntax

`validatePropertiesImpl(obj)`

Description

`validatePropertiesImpl(obj)` validates interdependent or interrelated property values at the beginning of object initialization, such as checking that the dependent or related inputs are the same size.

`validatePropertiesImpl` is the first method called by the `setup` method. `validatePropertiesImpl` also is called before the `processTunablePropertiesImpl` method.

Note You must set `Access=protected` for this method.

Input Arguments

obj

System object handle

Examples

Validate that the `useIncrement` property is `true` and that the value of the `increment` property is greater than zero.

```
methods (Access=protected)
   function validatePropertiesImpl(obj)
      if obj.useIncrement && obj.increment < 0
         error('The increment value must be positive');
      end
    end
end
```

`| | processTunedPropertiesImpl | | | | setupImpl | | | | validateInputsImpl`

-
-
- “Validate Property and Input Values”

matlab.system.mixin.FiniteSource

Purpose

Finite source mixin class

Description

`matlab.system.mixin.FiniteSource` is a class that defines the `isDone` method, which reports the state of a finite data source, such as an audio file.

To use this method, you must subclass from this class in addition to the `matlab.System` base class. You use the following syntax as the first line of your class definition file, where `ObjectName` is the name of your object:

```
classdef ObjectName < matlab.System &...
    matlab.system.mixin.FiniteSource
```

Methods

isDoneImpl	End-of-data flag

`| | matlab.System`

-
-
- "Define Finite Source Objects"

How To

- "Object-Oriented Programming"
- Class Attributes
- Property Attributes

Purpose

End-of-data flag

Syntax

`status = isDoneImpl(obj)`

Description

`status = isDoneImpl(obj)` indicates if an end-of-data condition has occurred. The `isDone` method should return `false` when data from a finite source has been exhausted, typically by having read and output all data from the source. You should also define the result of future reads from an exhausted source in the `isDoneImpl` method.

`isDoneImpl` is called by the `isDone` method.

Input Arguments

obj

System object handle

Output Arguments

status

Logical value, `true` or `false`, that indicates if an end-of-data condition has occurred or not, respectively.

Examples

Set up `isDoneImpl` so the `isDone` method checks whether the object has completed eight iterations.

```
methods (Access=private)
   function bdone = isDoneImpl(obj)
      bdone = obj.NumIters==8;
   end
end
```

`| | matlab.system.mixin.FiniteSource`

-
-
- “Define Finite Source Objects”

matlab.system.StringSet

Purpose

Set of valid string values

Description

`matlab.system.StringSet` defines a list of valid string values for a property. This class validates the string in the property and enables tab completion for the property value. A *StringSet* allows only predefined or customized strings as values for the property.

A `StringSet` uses two linked properties, which you must define in the same class. One is a public property that contains the current string value. This public property is displayed to the user. The other property is a hidden property that contains the list of all possible string values. This hidden property should also have the transient attribute so its value is not saved to disk when you save the System object.

The following considerations apply when using StringSets:

- The string property that holds the current string can have any name.
- The property that holds the `StringSet` must use the same name as the string property with the suffix "Set" appended to it. The string set property is an instance of the `matlab.system.StringSet` class.
- Valid strings, defined in the `StringSet`, must be declared using a cell array. The cell array cannot be empty nor can it have any empty strings. Valid strings must be unique and are case-insensitive.
- The string property must be set to a valid `StringSet` value.

Examples

Set the string property, `Flavor`, and the `StringSet` property, `FlavorSet`, in this example.

```
properties
  Flavor='Chocolate';
end

properties (Hidden,Transient)
  FlavorSet = ...
    matlab.system.StringSet({'Vanilla','Chocolate'});
end
```

| | `matlab.System`

How To

- “Object-Oriented Programming”
- Class Attributes
- Property Attributes
-
-
-
- “Limit Property Values to Finite String Set”

phased.ADPCACanceller

Purpose

Adaptive DPCA (ADPCA) pulse canceller

Description

The `ADPCACanceller` object implements an adaptive displaced phase center array pulse canceller.

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

1. Define and set up your ADPCA pulse canceller. See "Construction" on page 3-34.

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.

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 3-34 for the list of available property names.

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.

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`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency (PRF) of the received signal in hertz as a scalar.

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 (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'`.

Default: `[0; 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'`.

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 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.

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.

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

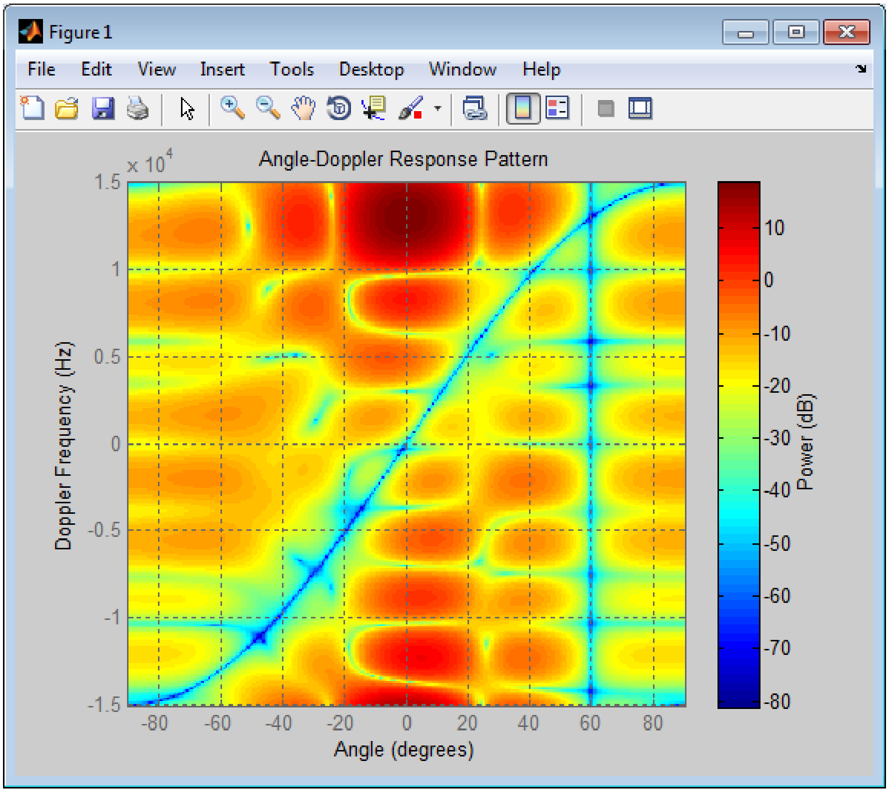
Methods

clone	Create ADPCA object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform ADPCA processing on input data

Examples

Process the data cube using an ADPCA processor. The weights are calculated for the 71st cell of a collected data cube. The look direction is [0 0] degrees and the Doppler is 12980 Hz.

```
load STAPExampleData;   % load radar data cube
Hs = 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] = step(Hs,STAPEx_ReceivePulse,71,[0; 0],12980);
Hresp = phased.AngleDopplerResponse(...
    'SensorArray',Hs.SensorArray,...
    'OperatingFrequency',Hs.OperatingFrequency,...
    'PRF',Hs.PRF,...
    'PropagationSpeed',Hs.PropagationSpeed);
plotResponse(Hresp,w);
```



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.

See Also

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

phased.ADPCACanceller.clone

Purpose Create ADPCA object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ADPCACanceller.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ADPCACanceller` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.ADPCACanceller.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform ADPCA processing on input data

Syntax

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

Description

`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 mainlobe direction. This syntax is available when the `DirectionSource` property is `'Input port'` and the `DopplerSource` property is `'Property'`.

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

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

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

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).

CUTIDX

Range cell.

ANG

Receiving mainlobe 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.

Default: `Direction` property of `H`

DOP

Targeting Doppler frequency in hertz. `DOP` must be a scalar.

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 the example radar data cube, `STAPExampleData.mat`, using an ADPCA processor. The weights are calculated for the 71st cell of a collected radar data cube. The look direction is `[0; 0]` degrees and the Doppler frequency is 12980 Hz. After constructing the `phased.ADPCACanceller` object, use `step` to process the data.

```
load STAPExampleData;   % load radar data cube
Hs = 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] = step(Hs,STAPEx_ReceivePulse,71,[0; 0],12980);
```

See Also

`uv2azel | phitheta2azel`

phased.AngleDopplerResponse

Purpose

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 3-48.

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.

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

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 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

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`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency (PRF) in hertz of the input signal as a positive scalar.

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'`.

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.

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.

Default: `256`

Methods

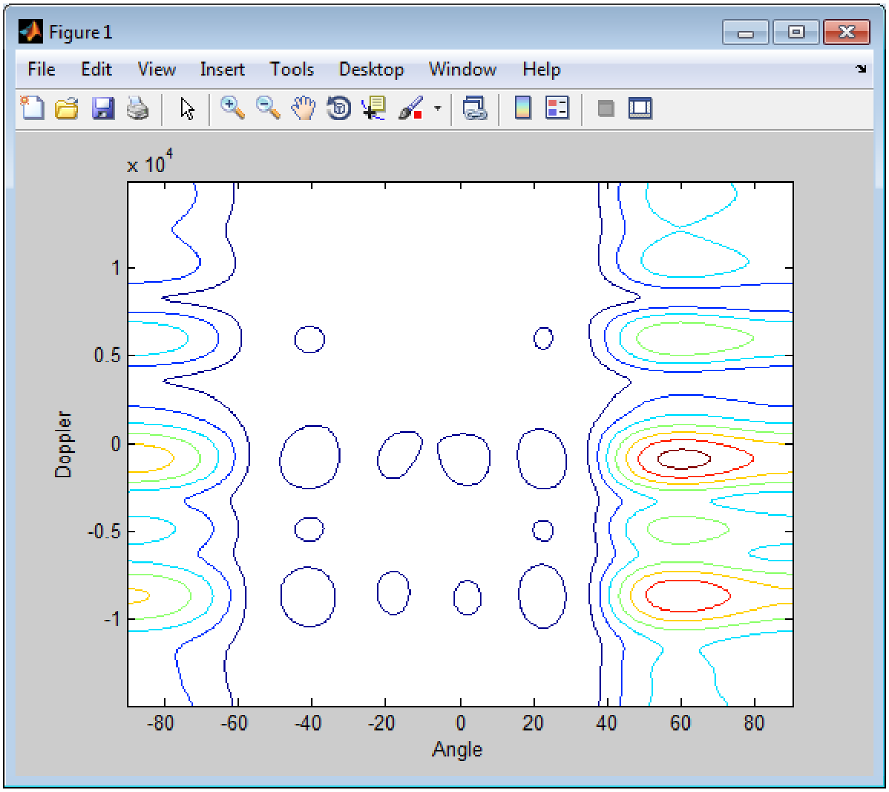
clone	Create angle-Doppler response object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties

plotResponse	Plot angle-Doppler response
release	Allow property value and input characteristics changes
step	Calculate angle-Doppler response

Examples

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

```
load STAPExampleData;
x = shiftdim(STAPEx_ReceivePulse(190,:,:));
% Construct angle-Doppler response object
hadresp = phased.AngleDopplerResponse(...
    'SensorArray',STAPEx_HArray,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'PRF',STAPEx_PRF);
% Use the step method to obtain the angle-Doppler response
[resp,ang_grid,dop_grid] = step(hadresp,x);
% Plot the angle-Doppler response
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

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

Purpose Create angle-Doppler response object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.AngleDopplerResponse.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`N = getNumInputs(H)`

Description

`N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.AngleDopplerResponse.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `AngleDopplerResponse` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.AngleDopplerResponse.plotResponse

Purpose

Plot angle-Doppler response

Syntax

```
plotResponse(H,X)
plotResponse(H,X,ELANG)
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'`.

`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(___,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`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

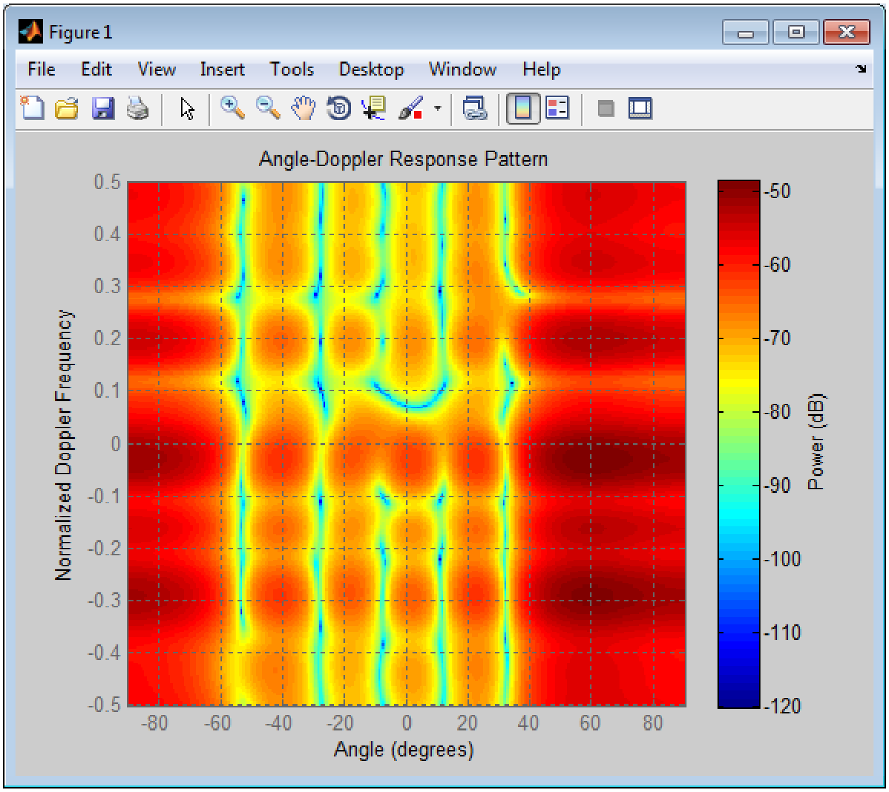
The unit of the plot. Valid values are `'db'`, `'mag'`, and `'pow'`.

Default: `'db'`

Examples

Plot the angle-Doppler response of 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`

phased.AngleDopplerResponse.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Calculate angle-Doppler response

Syntax

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

Description

`[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'`.

`[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'`.

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

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.

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

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

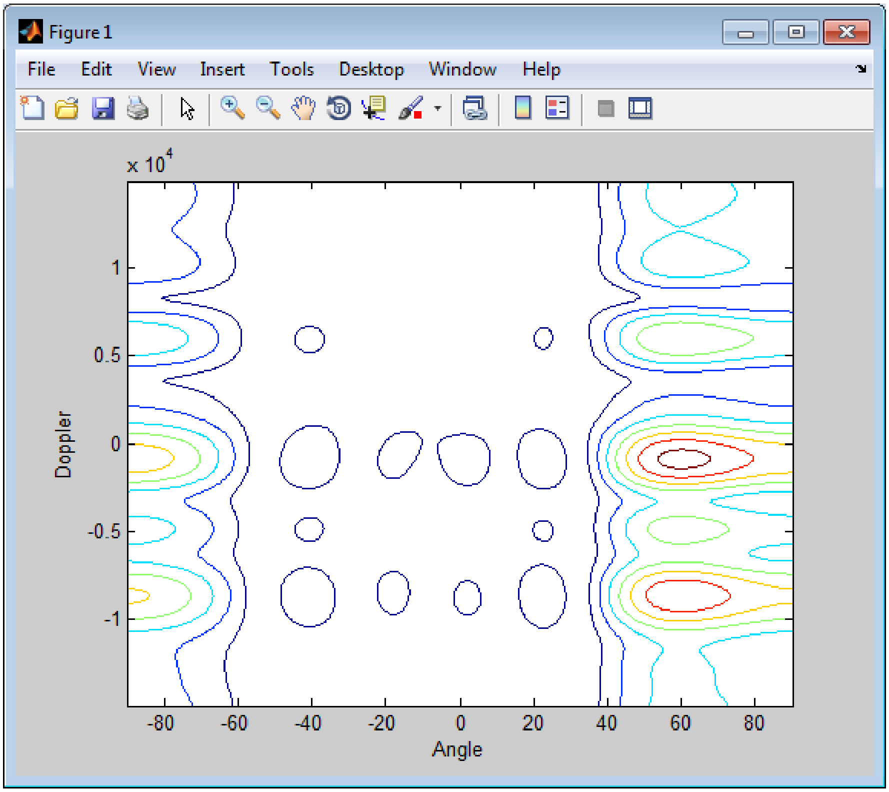
Doppler samples at which the angle-Doppler response is evaluated. `DOP_GRID` is a column vector of length P.

Examples

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

```
load STAPExampleData;
x = shiftdim(STAPEx_ReceivePulse(190,:,:));
% Construct angle-Doppler response object
hadresp = phased.AngleDopplerResponse(...
    'SensorArray',STAPEx_HArray,...
    'OperatingFrequency',STAPEx_OperatingFrequency,...
    'PropagationSpeed',STAPEx_PropagationSpeed,...
    'PRF',STAPEx_PRF);
% Use the step method to obtain the angle-Doppler response
[resp,ang_grid,dop_grid] = step(hadresp,x);
% Plot the angle-Doppler response
```

```
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

Purpose

Sensor array gain

Description

The `ArrayGain` object calculates the array gain for a sensor array. 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. 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 3-64.
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.

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

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 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

clone	Create array gain object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Calculate array gain of sensor array

Definitions

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:

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

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.

Examples

Calculate the array gain for a uniform linear array at the direction of 30 degrees azimuth and 20 degrees elevation. The array operating frequency is 300 MHz.

```
ha = phased.ULA(4);
hag = phased.ArrayGain('SensorArray',ha);
g = step(hag,3e8,[30;20]);
```

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.

See Also

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

phased.ArrayGain.clone

Purpose Create array gain object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ArrayGain.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ArrayGain` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.ArrayGain.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

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)
```

Description

`G = step(H,FREQ,ANG)` returns the array gain `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'`.

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

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.

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`.

Definitions

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:

$$\frac{SNR_{\text{out}}}{SNR_{\text{in}}} = \frac{\left(\dfrac{w^H vsv^H w}{w^H Nw}\right)}{\left(\dfrac{s}{N}\right)} = \frac{w^H vv^H w}{w^H w}$$

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.

Examples

Construct a uniform linear array with six elements. The array operates at 1 GHz and the array elements are spaced at one half the operating frequency wavelength. Find the array gain in decibels for the direction 45 degrees azimuth and 10 degrees elevation.

```
% operating frequency 1 GHz
fc = 1e9;
% 1 GHz wavelength
lambda = physconst('LightSpeed')/fc;
% construct the ULA
hULA = phased.ULA('NumElements',6,'ElementSpacing',lambda/2);
% construct the array gain object with the ULA as the sensor array
hgain = phased.ArrayGain('SensorArray',hULA);
% use step method to determine array gain at the specified
% operating frequency and angle
arraygain = step(hgain,fc,[45;10]);
% array gain is approximately -17.93 dB
```

See Also

`uv2azel | phitheta2azel`

Purpose

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 3-77.
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.

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`

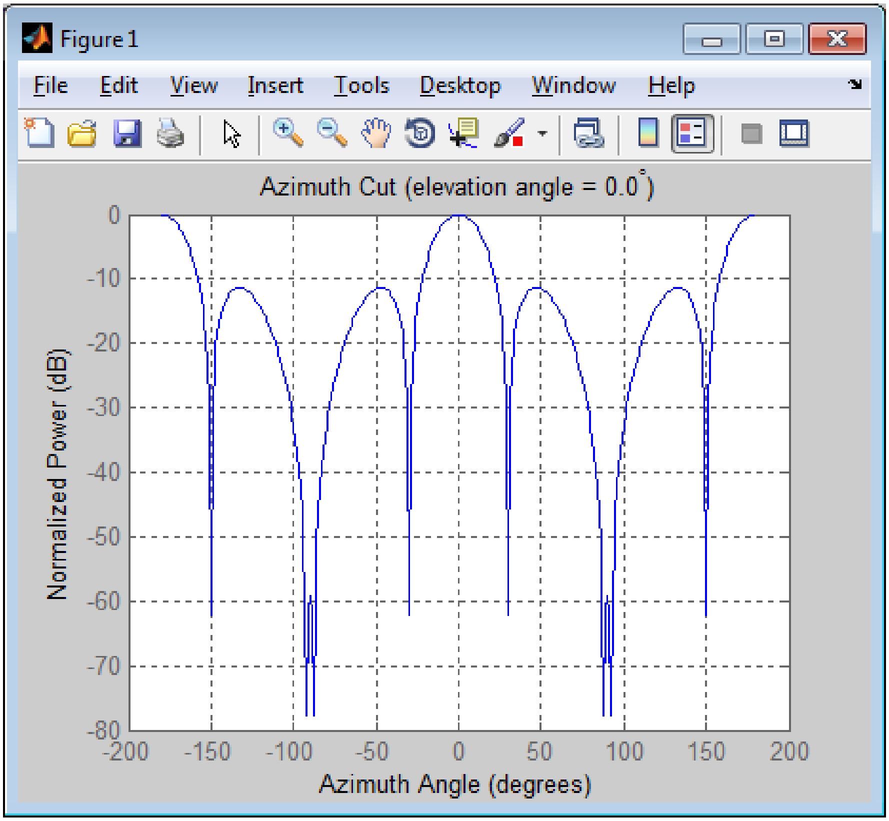
Methods

clone	Create array response object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Calculate array response of sensor array

Examples

Calculate the array response 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.

```
ha = phased.ULA(4);
har = phased.ArrayResponse('SensorArray',ha);
resp = step(har,3e8,[30;20]);
% Plot the array response in dB (azimuth cut--normalized power)
plotResponse(ha,3e8,physconst('LightSpeed'));
```



References

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

See Also

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

phased.ArrayResponse.clone

Purpose Create array response object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.ArrayResponse.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ArrayResponse.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ArrayResponse` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.ArrayResponse.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

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)
```

Description

`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'`.

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

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.

Output Arguments

RESP

Response of sensor array. `RESP` is an M-by-L matrix. `RESP` contains the array responses at the M angles specified in `ANG` and the L frequencies specified in `FREQ`.

Examples

Find the array response for a 6-element uniform linear array operating at 1 GHz. The array elements are spaced at one half the operating frequency wavelength. The incident angle is 45 degrees azimuth and 10 degrees elevation.

```
fc = 1e9;
% 1 GHz wavelength
lambda = physconst('LightSpeed')/fc;
% construct the ULA
hULA = phased.ULA('NumElements',6,'ElementSpacing',lambda/2);
% construct array response object with the ULA as sensor array
har = phased.ArrayResponse('SensorArray',hULA);
% use step to obtain array response at 1 GHz for an incident
% angle of 45 degrees azimuth and 10 degrees elevation
resp = step(har,fc,[45;10]);
```

See Also

`uv2azel | phitheta2azel`

phased.BarrageJammer

Purpose

Barrage jammer

Description

The `BarrageJammer` object implements a white Gaussian noise jammer.

To obtain the jamming signal:

1. Define and set up your barrage jammer. See "Construction" on page 3-88.
2. Call `step` to compute the jammer output according to the properties of `phased.BarrageJammer`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.BarrageJammer` creates a barrage jammer System object, `H`. This object generates a complex white Gaussian noise jamming signal.

`H = phased.BarrageJammer(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.BarrageJammer(E,Name,Value)` creates a barrage jammer object, `H`, with the `ERP` property set to `E` and other specified property Names set to the specified Values.

Properties

ERP

Effective radiated power

Specify the effective radiated power (ERP) (in watts) of the jamming signal as a positive scalar.

Default: `5000`

SamplesPerFrameSource

Source of number of samples per frame

Specify whether the number of samples of the jamming signal comes from the `SamplesPerFrame` property of this object or from an input argument in `step`. Values of this property are:

`'Property'`	The `SamplesPerFrame` property of this object specifies the number of samples of the jamming signal.
`'Input port'`	An input argument in each invocation of `step` specifies the number of samples of the jamming signal.

Default: `'Property'`

SamplesPerFrame

Number of samples per frame

Specify the number of samples in the output jamming signal as a positive integer. This property applies when you set the `SamplesPerFrameSource` property to `'Property'`.

Default: `100`

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

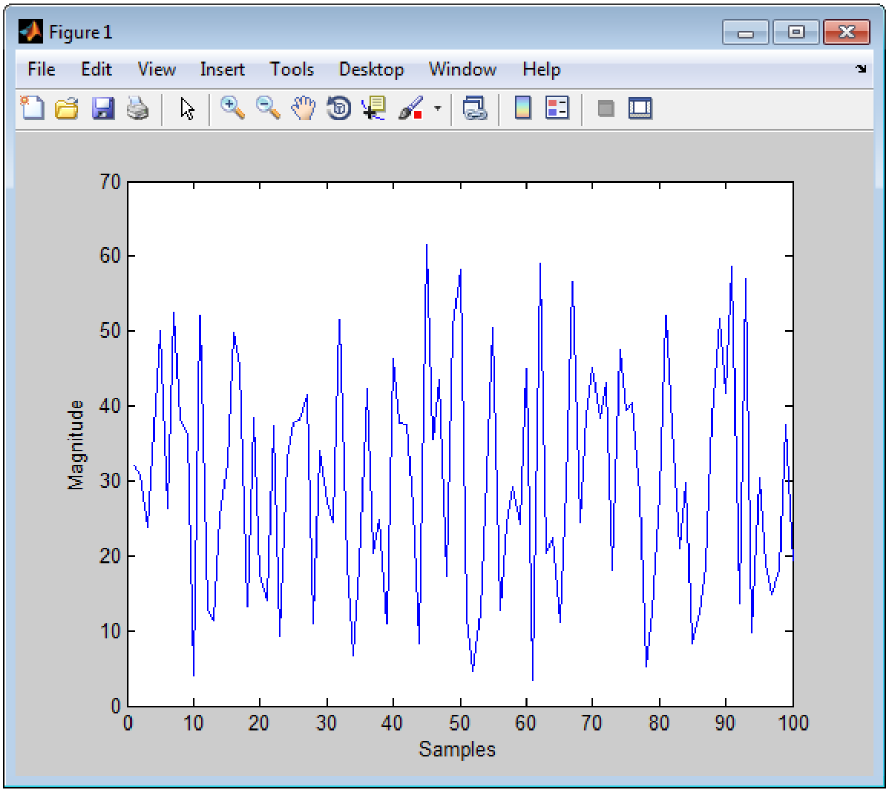
clone	Create barrage jammer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties

release	Allow property value and input characteristics changes
reset	Reset random number generator for noise generation
step	Generate noise jamming signal

Examples

Create a barrage jammer with an effective radiated power of 1000 w and plot the magnitude of that jammer's output. Your plot might vary because of random numbers.

```
Hjammer = phased.BarrageJammer('ERP',1000);
x = step(Hjammer);
plot(abs(x)); xlabel('Samples'); ylabel('Magnitude');
```



References

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

See Also

`phased.Platform` | `phased.RadarTarget` |

phased.BarrageJammer.clone

Purpose Create barrage jammer object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.BarrageJammer.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.BarrageJammer.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `BarrageJammer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.BarrageJammer.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.BarrageJammer.reset

Purpose Reset random number generator for noise generation

Syntax `reset(H)`

Description `reset(H)` resets the states of the `BarrageJammer` object, `H`. This method resets the random number generator state if the `SeedSource` property is set to `'Property'`.

Purpose

Generate noise jamming signal

Syntax

```
Y = step(H)
Y = step(H,N)
```

Description

`Y = step(H)` returns a column vector, `Y`, that is a complex white Gaussian noise jamming signal. The power of the jamming signal is specified by the `ERP` property. The length of the jamming signal is specified by the `SamplesPerFrame` property. This syntax is available when the `SamplesPerFrameSource` property is `'Property'`.

`Y = step(H,N)` returns the jamming signal with length `N`. This syntax is available when the `SamplesPerFrameSource` property is `'Input port'`.

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.

Examples

Create a barrage jammer with an effective radiated power of 1000 w and plot the magnitude of that jammer's output. Your plot might vary because of random numbers.

```
Hjammer = phased.BarrageJammer('ERP',1000);
x = step(Hjammer);
plot(abs(x)); xlabel('Samples'); ylabel('Magnitude');
```

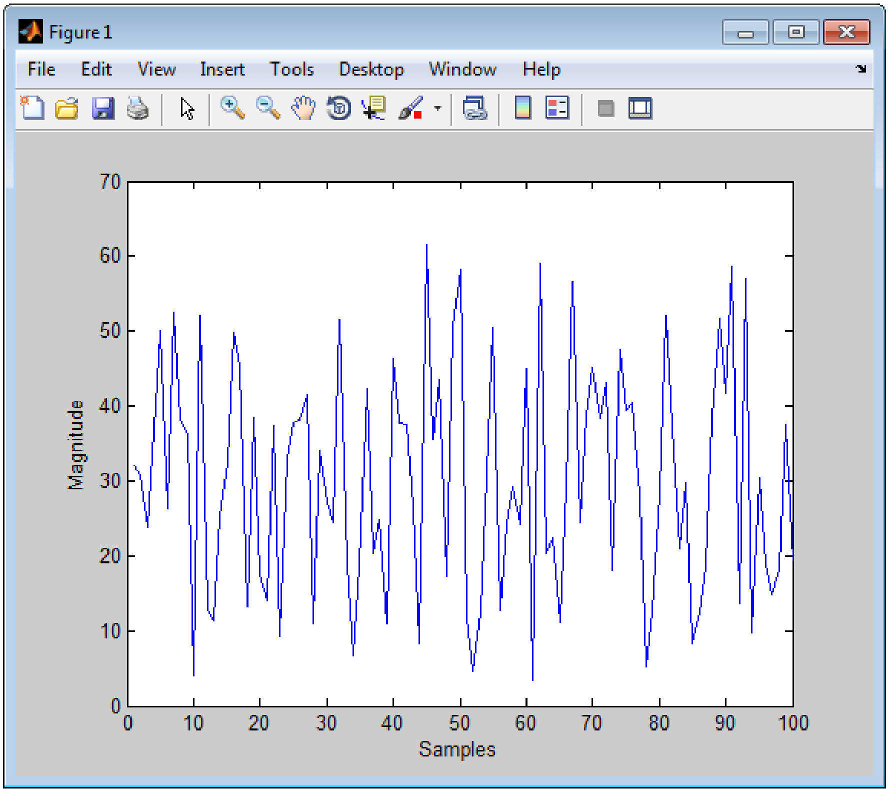


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Magnitude
Samples
0
10
20
30
40
50
60
70
0
10
20
30
40
50
60
70
80
90
100

Purpose

Beamscan spatial spectrum estimator for ULA

Description

The `BeamscanEstimator` object calculates a beamscan spatial spectrum estimate for a uniform linear array.

To estimate the spatial spectrum:

1. Define and set up your beamscan spatial spectrum estimator. See "Construction" on page 3-101.
2. Call `step` to estimate the spatial spectrum according to the properties of `phased.BeamscanEstimator`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.BeamscanEstimator` creates a beamscan spatial spectrum estimator System object, `H`. The object estimates the incoming signal's spatial spectrum using a narrowband conventional beamformer for a uniform linear array (ULA).

`H = phased.BeamscanEstimator(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`

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 elements 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

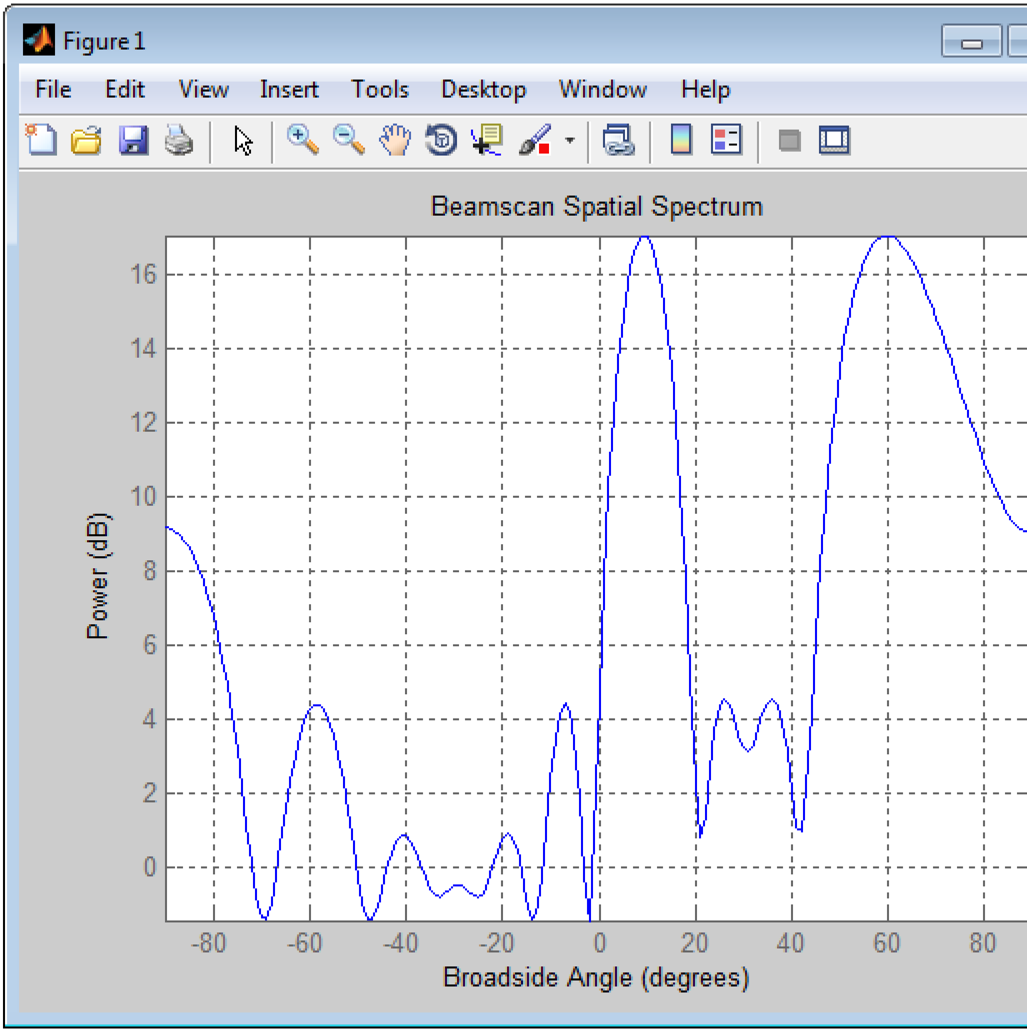
clone	Create beamscan spatial spectrum estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotSpectrum	Plot spatial spectrum
release	Allow property value and input characteristics changes

reset	Reset states of beamscan spatial spectrum estimator object
step	Perform spatial spectrum estimation

Examples

Estimate the DOAs of two signals received by a standard 10-element ULA with an element spacing of one 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 60 degrees in azimuth and –5 degrees in elevation. This example also plots the spatial spectrum.

```
fs = 8000; t = (0:1/fs:1).';
x1 = cos(2*pi*t*300); x2 = cos(2*pi*t*400);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;60 -5]',fc);
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
hdoa = phased.BeamscanEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2);
[y,doas] = step(hdoa,x+noise);
doas = broadside2az(sort(doas),[20 -5]);
plotSpectrum(hdoa);
```



References

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

See Also

`broadside2az` | `phased.BeamscanEstimator2D` |

phased.BeamscanEstimator.clone

Purpose Create beamscan spatial spectrum estimator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.BeamscanEstimator.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.BeamscanEstimator.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `BeamscanEstimator` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.BeamscanEstimator.plotSpectrum

Purpose

Plot spatial spectrum

Syntax

```
plotSpectrum(H)
plotSpectrum(H,Name,Value)
h = plotSpectrum(___)
```

Description

`plotSpectrum(H)` plots the spatial spectrum resulting from the last call of the `step` method.

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

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

Input Arguments

H

Spatial spectrum estimator object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

NormalizeResponse

Set this value to `true` to plot the normalized spectrum. Set this value to `false` to plot the spectrum without normalizing it.

Default: `false`

Title

String to use as title of figure.

Default: Empty string

Unit

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

Default: `'db'`

Examples

Estimate the DOAs 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 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 60 degrees in azimuth and –5 degrees in elevation.

```
fs = 8000; t = (0:1/fs:1).';
x1 = cos(2*pi*t*300); x2 = cos(2*pi*t*400);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;60 -5]',fc);
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
hdoa = phased.BeamscanEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2);
[y,doas] = step(hdoa,x+noise);
doas = broadside2az(sort(doas),[20 -5]);
plotSpectrum(hdoa);
```

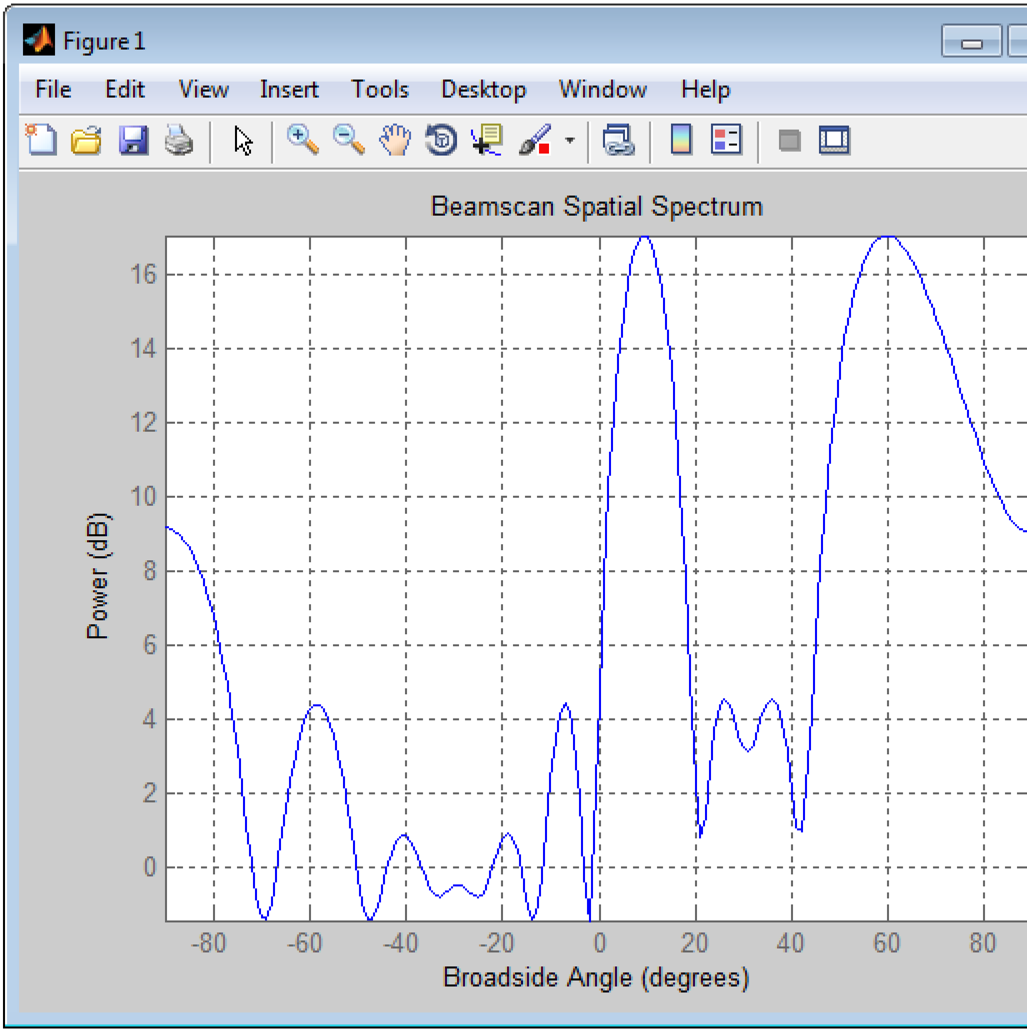


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Beamscan Spatial Spectrum
Power (dB)
16
14
12
10
8
6
4
2
0
-80
-60
-40
-20
0
20
40
60
80
Broadside Angle (degrees)

phased.BeamscanEstimator.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose Reset states of beamscan spatial spectrum estimator object

Syntax `reset(H)`

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

phased.BeamscanEstimator.step

Purpose Perform spatial spectrum estimation

Syntax

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

Description

`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.

`[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 `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.

Examples

Estimate the DOAs 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 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 60 degrees in azimuth and -5 degrees in elevation.

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

```
hdoa = phased.BeamscanEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2);
[y,doas] = step(hdoa,x+noise);
doas = broadside2az(sort(doas),[20 -5]);
```

See Also

`azel2uv` | `azel2phitheta`

phased.BeamscanEstimator2D

Purpose

2-D beamscan spatial spectrum estimator

Description

The `BeamscanEstimator2D` object calculates a 2-D beamscan spatial spectrum estimate.

To estimate the spatial spectrum:

1. Define and set up your 2-D beamscan spatial spectrum estimator. See "Construction" on page 3-118.
2. Call `step` to estimate the spatial spectrum according to the properties of `phased.BeamscanEstimator2D`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.BeamscanEstimator2D` creates a 2-D beamscan spatial spectrum estimator System object, `H`. The object estimates the signal's spatial spectrum using a narrowband conventional beamformer.

`H = phased.BeamscanEstimator2D(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.

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`

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

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.

Default: `-90:90`

ElevationScanAngles

Elevation scan angles

Specify the elevation scan angles (in degrees) as a real vector or scalar. The angles must be within [–90 90]. You must specify the angles in an ascending order.

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`.

Default: `1`

Methods

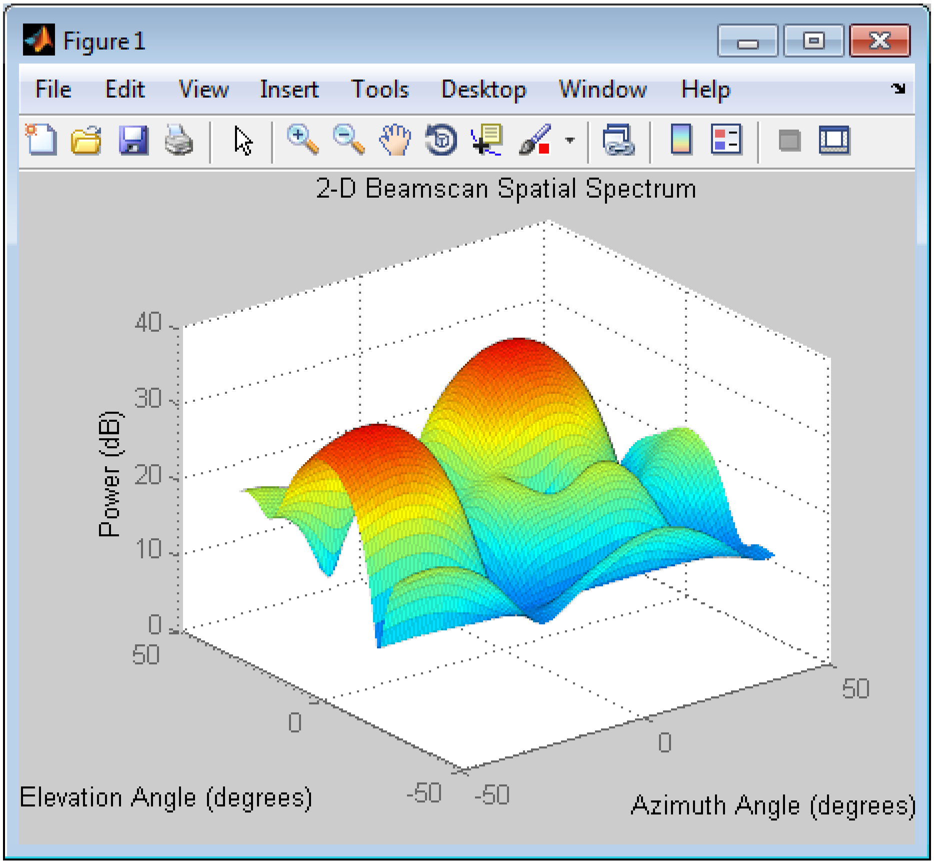
clone	Create 2-D beamscan spatial spectrum estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotSpectrum	Plot spatial spectrum
release	Allow property value and input characteristics changes

reset	Reset states of 2-D beamscan spatial spectrum estimator object
step	Perform spatial spectrum estimation

Examples

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 degrees in azimuth and 0 degrees in elevation. The direction of the second signal is 17 degrees in azimuth and 20 degrees in elevation. This example also plots the spatial spectrum.

```
ha = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
lambda = physconst('LightSpeed')/fc;
ang1 = [-37; 0]; ang2 = [17; 20];
x = sensorsig(getElementPosition(ha)/lambda,8000,[ang1 ang2],0.2);
hdoa = phased.BeamscanEstimator2D('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
[~,doas] = step(hdoa,x);
plotSpectrum(hdoa);
```



References

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

See Also

`phased.BeamscanEstimator` | `uv2azel` | `phitheta2azel`

Purpose

Create 2-D beamscan spatial spectrum estimator object with same property values

Syntax

`C = clone(H)`

Description

`C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.BeamscanEstimator2D.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.BeamscanEstimator2D.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.BeamscanEstimator2D.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `BeamscanEstimator2D` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose

Plot spatial spectrum

Syntax

```
plotSpectrum(H)
plotSpectrum(H,Name,Value)
h = plotSpectrum( ___ )
```

Description

`plotSpectrum(H)` plots the spatial spectrum resulting from the last call of the `step` method.

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

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

Input Arguments

H

Spatial spectrum estimator object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

NormalizeResponse

Set this value to `true` to plot the normalized spectrum. Set this value to `false` to plot the spectrum without normalizing it.

Default: `false`

Title

String to use as title of figure.

Default: Empty string

Unit

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

Default: `'db'`

Examples

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 degrees in azimuth and 0 degrees in elevation. The direction of the second signal is 17 degrees in azimuth and 20 degrees in elevation.

```
ha = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
lambda = physconst('LightSpeed')/fc;
ang1 = [-37; 0]; ang2 = [17; 20];
x = sensorsig(getElementPosition(ha)/lambda,8000,[ang1 ang2],0.2);
hdoa = phased.BeamscanEstimator2D('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
[~,doas] = step(hdoa,x);
plotSpectrum(hdoa);
```

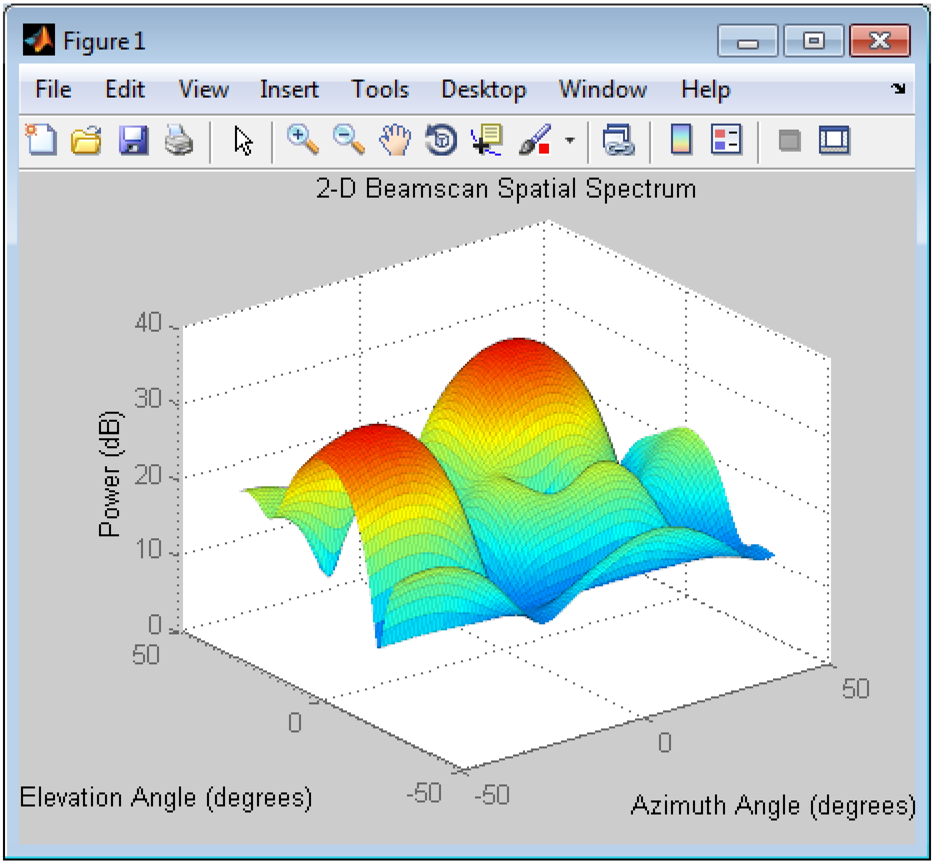


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
2-D Beamscan Spatial Spectrum
Power (dB)
40
30
20
10
0
50
0
-50
-50
0
50
Elevation Angle (degrees)
Azimuth Angle (degrees)

phased.BeamscanEstimator2D.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.BeamscanEstimator2D.reset

Purpose Reset states of 2-D beamscan spatial spectrum estimator object

Syntax `reset(H)`

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

phased.BeamscanEstimator2D.step

Purpose

Perform spatial spectrum estimation

Syntax

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

Description

`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`.

`[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).

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.

Examples

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 degrees in azimuth and 0 degrees in elevation. The direction of the second signal is 17 degrees in azimuth and 20 degrees in elevation.

```
ha = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
```

```
lambda = physconst('LightSpeed')/fc;
ang1 = [-37; 0]; ang2 = [17; 20];
x = sensorsig(getElementPosition(ha)/lambda,8000,[ang1 ang2],0.2);
hdoa = phased.BeamscanEstimator2D('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
[~,doas] = step(hdoa,x);
```

See Also

`azel2uv` | `azel2phitheta`

phased.BeamspaceESPRITEstimator

Purpose

Beamspace ESPRIT direction of arrival (DOA) estimator

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 3-134.
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.

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.

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`

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

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 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'`.

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. 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'`.

Default: `2`

Methods

clone	Create beamspace ESPRIT DOA estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform DOA estimation

Examples

Estimate the DOAs 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 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);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
% construct beamspace ESPRIT estimator
hdoa = phased.BeamspaceESPRITEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property','NumSignals',2);
% use the step method to obtain the direction of arrival estimates
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60]);
```

References

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

See Also

`broadside2azphased.ESPRITEstimator` |

Purpose Create beamspace ESPRIT DOA estimator object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.BeamspaceESPRITEstimator.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`N = getNumInputs(H)`

Description

`N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.BeamspaceESPRITEstimator.getNumOutputs

Purpose Number of outputs from step method

Syntax

```
N = getNumOutputs(H)
```

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.BeamspaceESPRITEstimator.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `BeamspaceESPRITEstimator` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.BeamspaceESPRITEstimator.release

Purpose

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.BeamspaceESPRITEstimator.step

Purpose

Perform DOA estimation

Syntax

`ANG = step(H,X)`

Description

`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).

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.

Examples

Estimate the DOAs 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 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);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
% construct beamspace ESPRIT estimator
hdoa = phased.BeamspaceESPRITEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property','NumSignals',2);
% use the step method to obtain the direction of arrival estimates
```

```
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60]);
```

Purpose

Constant false alarm rate (CFAR) detector

Description

The `CFARDetector` object implements a constant false-alarm rate detector.

To perform the detection:

1. Define and set up your CFAR detector. See "Construction" on page 3-146.
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.

Construction

`H = phased.CFARDetector` creates a constant false alarm rate (CFAR) detector System object, `H`. The object performs CFAR detection on the input data.

`H = phased.CFARDetector(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

Method

CFAR algorithm

Specify the algorithm of the CFAR detector as a string. Values of this property are:

`'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'`.

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.

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.

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.

Default: `1`

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`

Methods

clone	Create CFAR detector object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform CFAR detection

Examples

Perform cell-averaging CFAR detection on a given Gaussian noise vector with a desired probability of false alarm of 0.1. Assume that the data is 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 input.

```
rng(5);
hdet = phased.CFARDetector('NumTrainingCells',50,...
    'NumGuardCells',2,'ProbabilityFalseAlarm',0.1);
```

```
N = 1000; x = 1/sqrt(2)*(randn(N,1)+1i*randn(N,1));
dresult = step(hdet,abs(x).^2,1:N);
Pfa = sum(dresult)/N;
```

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 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.

For further details, see [1].

References

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

See Also

`npwgnthreshphased.MatchedFilter` **|** `phased.TimeVaryingGain` **|**

Purpose Create CFAR detector object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.CFARDetector.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.CFARDetector.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `CFARDetector` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose Allow property value and input characteristics changes

Syntax

```
release(H)
```

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

phased.CFARDetector.step

Purpose Perform CFAR detection

Syntax

```
Y = step(H,X,CUTIDX)
Y = step(H,X,CUTIDX,THFAC)
[Y,TH] = step(___)
```

Description

`Y = step(H,X,CUTIDX)` performs the CFAR detection on the real input data `X`. `X` can be either a column vector or a matrix. Each row of `X` is a cell and each column of `X` is independent data. Detection is performed along each column for the cells specified in `CUTIDX`. `CUTIDX` must be a vector of positive integers with each entry specifying the index of a cell under test (CUT). `Y` is an M-by-N matrix containing the logical detection result for the cells in `X`. M is the number of indices specified in `CUTIDX`, and N is the number of independent signals in `X`.

`Y = step(H,X,CUTIDX,THFAC)` uses `THFAC` as the threshold factor used to calculate the detection threshold. This syntax is available when you set the `ThresholdFactor` property to `'Input port'`. `THFAC` must be a positive scalar.

`[Y,TH] = step(___)` returns additional output, `TH`, as the detection threshold for each cell under test in `X`. This syntax is available when you set the `ThresholdOutputPort` property to `true`. `TH` has the same dimensionality as `Y`.

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.

Examples

Perform cell-averaging CFAR detection on a given Gaussian noise vector with a desired probability of false alarm of 0.1. Assume that the data is 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 input.

```
rng(5);
hdet = phased.CFARDetector('NumTrainingCells',50,...
    'NumGuardCells',2,'ProbabilityFalseAlarm',0.1);
N = 1000; x = 1/sqrt(2)*(randn(N,1)+1i*randn(N,1));
dresult = step(hdet,abs(x).^2,1:N);
Pfa = sum(dresult)/N;
```

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.

Purpose

Narrowband signal collector

Description

The `Collector` object implements a narrowband signal collector.

To compute the collected signal at the sensor(s):

1. Define and set up your signal collector. See "Construction" on page 3-159.
2. Call `step` to collect the signal according to the properties of `phased.Collector`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.Collector` creates a narrowband signal collector System object, `H`. The object collects incident narrowband signals from given directions using a sensor array or a single element.

`H = phased.Collector(Name,Value)` creates a collector 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

Sensor

Handle of sensor

Specify the sensor as a sensor array object or an element object in the `phased` package. If the sensor is an array, it 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

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`

WeightsInputPort

Enable weights input

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`

Wavefront

Type of incoming wavefront

Specify the type of incoming wavefront as one of `'Plane'`, or `'Unspecified'`:

- If you set the `Wavefront` property to `'Plane'`, the input signals are multiple plane waves impinging on the entire array. Each plane wave is received by all collecting elements. If the `Sensor` property is an array that contains subarrays, the `Wavefront` property must be `'Plane'`.
- If you set the `Wavefront` property to `'Unspecified'`, the input signals are individual waves impinging on individual sensors.

Default: `'Plane'`

Methods

clone	Create collector object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Collect signals

Examples

Collect signal with a single antenna.

```
ha = phased.IsotropicAntennaElement;
hc = phased.Collector('Sensor',ha,'OperatingFrequency',1e9);
x = [1;1];
incidentAngle = [10 30]';
y = step(hc,x,incidentAngle);
```

Collect a far field signal with a 5-element array.

```
ha = phased.ULA('NumElements',5);
hc = phased.Collector('Sensor',ha,'OperatingFrequency',1e9);
x = [1;1];
incidentAngle = [10 30]';
y = step(hc,x,incidentAngle);
```

Collect signals with a 3-element array. Each antenna collects a separate input signal from a separate direction.

```
ha = phased.ULA('NumElements',3);
hc = phased.Collector('Sensor',ha,'OperatingFrequency',1e9,...
    'Wavefront','Unspecified');
x = rand(10,3);   % Each column is a separate signal for one element
incidentAngle = [10 0; 20 5; 45 2]'; % 3 angles for 3 signals
y = step(hc,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

`phased.WidebandCollector` |

Purpose Create collector object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.Collector.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax

```
N = getNumOutputs(H)
```

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.Collector.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `Collector` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.Collector.step

Purpose

Collect signals

Syntax

```
Y = step(H,X,ANG)
Y = step(H,X,ANG,WEIGHTS)
Y = step(H,X,ANG,STEERANGLE)
Y = step(H,X,ANG,WEIGHTS,STEERANGLE)
```

Description

`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,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,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 `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

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.

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.

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

Construct a 4-element uniform linear array. The array operating frequency is 1 GHz. The array element spacing is half the operating frequency wavelength. Model the collection of a 200-Hz sine wave incident on the array from 45 degrees azimuth, 10 degrees elevation from the far field.

```
fc = 1e9;
lambda = physconst('LightSpeed')/fc;
hULA = phased.ULA('NumElements',4,'ElementSpacing',lambda/2);
t = linspace(0,1,1e3);
x = cos(2*pi*200*t)';
% construct the collector object.
hc = phased.Collector('Sensor',hULA,...
    'PropagationSpeed',physconst('LightSpeed'),...
    'Wavefront','Plane','OperatingFrequency',fc);
% incident angle is 45 degrees azimuth, 10 degrees elevation
```

```
incidentangle = [45;10];
% collect the incident waveform at the ULA
receivedsig = step(hc,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

Purpose

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 3-172.

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.

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. To specify a value-only argument, you must also specify all preceding value-only arguments. You can specify name-value arguments in any order.

Properties

Element

Element of array

Specify the element of the sensor array as a handle. The element must be an element object in the `phased` package.

Default: An isotropic antenna element that operates between 300 MHz and 1 GHz

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 array's local coordinate system. 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. `ElementNormal` must be a 2-by-N matrix, where N indicates the number of elements in the array. Each column of `ElementNormal` specifies the normal direction of the corresponding element in the form [azimuth; elevation] (in degrees) defined in the local coordinate system. The local coordinate system aligns the positive *x*-axis with the direction normal to the conformal 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]`

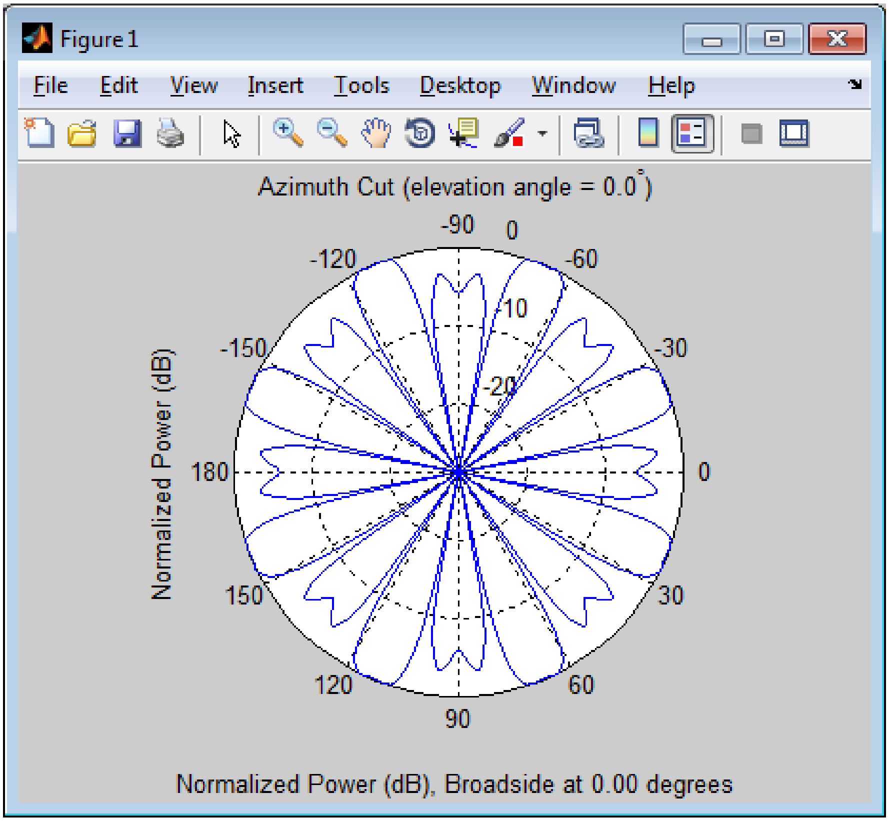
Methods

clone	Create conformal array object with same property values
collectPlaneWave	Simulate received plane waves
getElementPosition	Positions of array elements
getNumElements	Number of elements in array
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of array
release	Allow property value and input characteristics changes
step	Output responses of array elements
viewArray	View array geometry

Examples

Construct an 8-element uniform circular array (UCA) and plot its azimuth responses. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

```
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)]);
fc = 1e9; c = 3e8;
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



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.

See Also

`phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.URA` | `uv2azel` | `phitheta2azel`

Related Examples

- Phased Array Gallery

Purpose

Create conformal array object with same property values

Syntax

```
C = clone(H)
```

Description

`C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.ConformalArray.collectPlaneWave

Purpose

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)` uses `FREQ` as the incoming signal's carrier frequency.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)` uses `C` as the signal's propagation speed. `C` must be a scalar.

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 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, 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 the received signal at an 8-element uniform circular array.

The signals arrive from 10 degrees 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.

```
N = 8; azang = (0:N-1)*360/N-180;
hArray = phased.ConformalArray(...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
y = collectPlaneWave(hArray,randn(4,2),[10 30],1e8);
```

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`

Purpose 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 Construct a default conformal array and obtain the element positions.

```
ha = phased.ConformalArray;
pos = getElementPosition(ha)
```

phased.ConformalArray.getNumElements

Purpose 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 Construct a default conformal array and obtain the number of elements.

```
ha = phased.ConformalArray;
N = getNumElements(ha)
```

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ConformalArray.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `ConformalArray` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.ConformalArray.plotResponse

Purpose

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.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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can

specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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'`, and `'pow'`.

Default: `'db'`

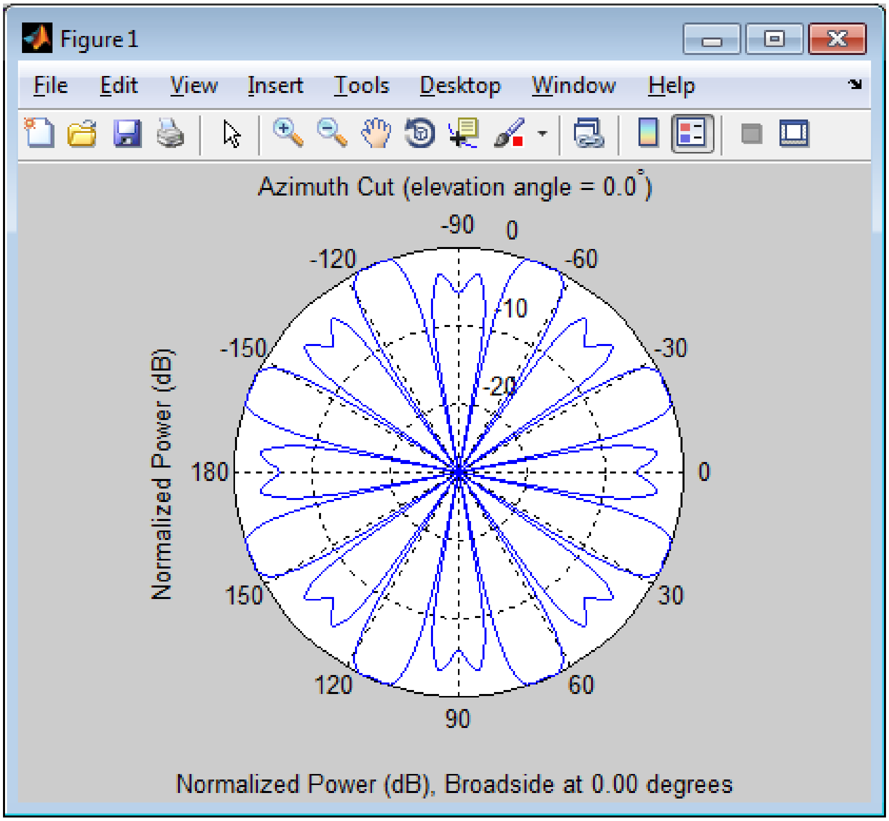
Weights

Weights applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. *N* is the number of elements in the array. *M* is the number of frequencies in `FREQ`. If `Weights` is a vector, the function applies the same weights to each frequency. If `Weights` is a matrix, the function applies each column of weight values to the corresponding frequency in `FREQ`.

Examples

Construct an 8-element uniform circular array (UCA) and plot its azimuth responses. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

```
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)]);
fc = 1e9; c = 3e8;
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



See Also uv2azel | azel2uv

phased.ConformalArray.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

`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 `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

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` 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

Responses of array elements. `RESP` has dimensions N-by-M-by-L. N is the number of elements in the phased array. 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

Construct an 8-element uniform circular array (UCA). Assume the operating frequency is 1 GHz. Find the response of each element in this array in the direction of 30 degrees azimuth and 5 degrees elevation.

```
ha = phased.ConformalArray;
N = 8; azang = (0:N-1)*360/N-180;
ha.ElementPosition = [cosd(azang);sind(azang);zeros(1,N)];
ha.ElementNormal = [azang;zeros(1,N)];
fc = 1e9; ang = [30;5];
resp = step(ha,fc,ang);
```

See Also

`uv2azel` | `phitheta2azel`

Purpose

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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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 string `'All'` to show indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

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`

Title

String specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

Handle of array elements in figure window.

Examples

Positions and Normal Directions in Uniform Circular Array

Display the element positions and normal directions of all elements of an 8-element uniform circular array.

Create a vector of eight uniformly spaced azimuth angles.

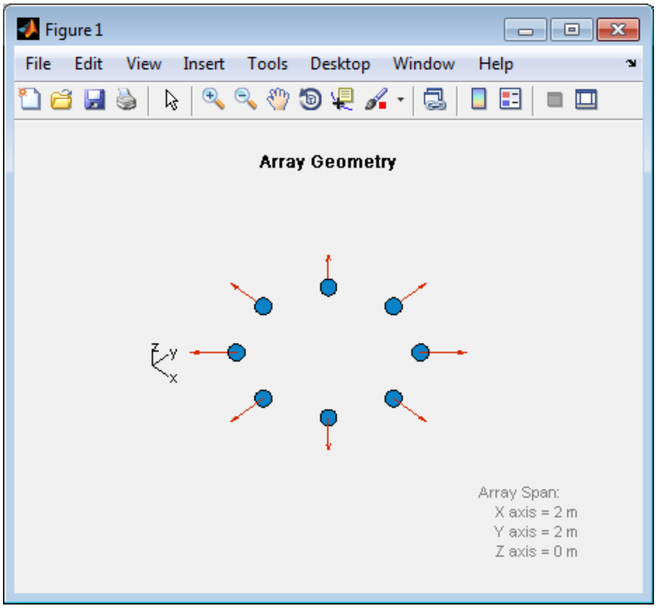
```
N = 8;
azang = (0:N-1) * 360/N - 180;
```

Create an 8-element uniform circular array.

```
ha = phased.ConformalArray(...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
```

Display the element positions and normal directions of all elements in the array.

```
viewArray(ha,'ShowNormals',true)
```



See Also `phased.ArrayResponse` |

Related Examples

- Phased Array Gallery

Purpose

Constant gamma clutter simulation

Description

The `ConstantGammaClutter` object simulates clutter.

To compute the clutter return:

1. Define and set up your clutter simulator. See "Construction" on page 3-196.
2. Call `step` to simulate the clutter return for your system according to the properties of `phased.ConstantGammaClutter`. The behavior of `step` is specific to each object in the toolbox.

The clutter simulation that `ConstantGammaClutter` provides is based on these assumptions:

- The radar system is monostatic.
- The propagation is in free space.
- The terrain is homogeneous.
- The clutter patch is stationary during the coherence time. *Coherence time* indicates how frequently the software changes the set of random numbers in the clutter simulation.
- The signal is narrowband. Thus, the spatial response can be approximated by a phase shift. Similarly, the Doppler shift can be approximated by a phase shift.
- The radar system maintains a constant height during simulation.
- The radar system maintains a constant speed during simulation.

Construction

`H = phased.ConstantGammaClutter` creates a constant gamma clutter simulation System object, `H`. This object simulates the clutter return of a monostatic radar system using the constant gamma model.

`H = phased.ConstantGammaClutter(Name,Value)` creates a constant gamma clutter simulation object, `H`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name, 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

Sensor

Handle of sensor

Specify the sensor as an antenna element object or as an array object whose `Element` property value is an antenna element object. If the sensor is an array, it 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

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`

SampleRate

Sample rate

Specify the sample rate, in hertz, as a positive scalar. The default value corresponds to 1 MHz.

Default: `1e6`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency in hertz as a positive scalar or a row vector. The default value of this property corresponds to 10 kHz. When `PRF` is a vector, it represents a staggered PRF. In this case, the output pulses use elements in the vector as their PRFs, one after another, in a cycle.

Default: `1e4`

Gamma

Terrain gamma value

Specify the γ value used in the constant γ clutter model, as a scalar in decibels. The γ value depends on both terrain type and the operating frequency.

Default: `0`

EarthModel

Earth model

Specify the earth model used in clutter simulation as one of `| 'Flat' | 'Curved' |`. When you set this property to `'Flat'`, the earth is assumed to be a flat plane. When you set this property to `'Curved'`, the earth is assumed to be a sphere.

Default: `'Flat'`

PlatformHeight

Radar platform height from surface

Specify the radar platform height (in meters) measured upward from the surface as a nonnegative scalar.

Default: `300`

PlatformSpeed

Radar platform speed

Specify the radar platform’s speed as a nonnegative scalar in meters per second.

Default: `300`

PlatformDirection

Direction of radar platform motion

Specify the direction of radar platform motion as a 2-by-1 vector in the form [AzimuthAngle; ElevationAngle] in degrees. The default value of this property indicates that the platform moves perpendicular to the radar antenna array’s broadside.

Both azimuth and elevation angle are measured in the local coordinate system of the radar antenna or antenna array. Azimuth angle must be between –180 and 180 degrees. Elevation angle must be between –90 and 90 degrees.

Default: `[90;0]`

BroadsideDepressionAngle

Depression angle of array broadside

Specify the depression angle in degrees of the broadside of the radar antenna array. This value is a scalar. The broadside is defined as zero degrees azimuth and zero degrees elevation. The depression angle is measured downward from horizontal.

Default: `0`

MaximumRange

Maximum range for clutter simulation

Specify the maximum range in meters for the clutter simulation as a positive scalar. The maximum range must be greater than the value specified in the `PlatformHeight` property.

Default: `5000`

AzimuthCoverage

Azimuth coverage for clutter simulation

Specify the azimuth coverage in degrees as a positive scalar. The clutter simulation covers a region having the specified azimuth span, symmetric to 0 degrees azimuth. Typically, all clutter patches have their azimuth centers within the region, but the `PatchAzimuthWidth` value can cause some patches to extend beyond the region.

Default: `60`

PatchAzimuthWidth

Azimuth span of each clutter patch

Specify the azimuth span of each clutter patch in degrees as a positive scalar.

Default: `1`

TransmitSignalInputPort

Add input to specify transmit signal

Set this property to `true` to add input to specify the transmit signal in the `step` syntax. Set this property to `false` omit the transmit signal in the `step` syntax. The `false` option is less computationally expensive; to use this option, you must also specify the `TransmitERP` property.

Default: `false`

TransmitERP

Effective transmitted power

Specify the transmitted effective radiated power (ERP) of the radar system in watts as a positive scalar. This property applies only when you set the `TransmitSignalInputPort` property to `false`.

Default: `5000`

CoherenceTime

Clutter coherence time

Specify the coherence time in seconds for the clutter simulation as a positive scalar. After the coherence time elapses, the `step` method updates the random numbers it uses for the clutter simulation at the next pulse. A value of `inf` means the random numbers are never updated.

Default: `inf`

OutputFormat

Output signal format

Specify the format of the output signal as one of `| 'Pulses' | 'Samples' |`. When you set the `OutputFormat` property to `'Pulses'`, the output of the `step` method is in the form of multiple pulses. In this case, the number of pulses is the value of the `NumPulses` property.

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. In staggered PRF applications, you might find the `'Samples'` option more convenient because the `step` output always has the same matrix size.

Default: `'Pulses'`

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`

NumSamples

Number of samples in output

Specify the number of samples in the output of the `step` method as a positive integer. Typically, you use the number of samples in one pulse. This property applies only when you set the `OutputFormat` property to `'Samples'`.

Default: `100`

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

clone	Create constant gamma clutter simulation object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties

release	Allow property value and input characteristics changes
reset	Reset random numbers and time count for clutter simulation
step	Simulate clutter using constant gamma model

Examples

Clutter Simulation of System with Known Power

Simulate the clutter return from terrain with a gamma value of 0 dB. The effective transmitted power of the radar system is 5 kw.

Set up the characteristics of the radar system. This system has a 4-element uniform linear array (ULA). The sample rate is 1 MHz, and the PRF is 10 kHz. The propagation speed is 300,000 km/s, and the operating frequency is 300 MHz. The radar platform is flying 1 km above the ground with a path parallel to the ground along the array axis. The platform speed is 2000 m/s. The mainlobe has a depression angle of 30 degrees.

```
Nele = 4;
c = 3e8; fc = 3e8; lambda = c/fc;
ha = phased.ULA('NumElements',Nele,'ElementSpacing',lambda/2);

fs = 1e6; prf = 10e3;
height = 1000; direction = [90; 0];
speed = 2000; depang = 30;
```

Create the clutter simulation object. The configuration assumes the earth is flat. The maximum clutter range of interest is 5 km, and the maximum azimuth coverage is +/– 60 degrees.

```
Rmax = 5000; Azcov = 120;
tergamma = 0; tpower = 5000;
hclutter = phased.ConstantGammaClutter('Sensor',ha,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'PRF',prf,...
    'SampleRate',fs,'Gamma',tergamma,'EarthModel','Flat',...
```

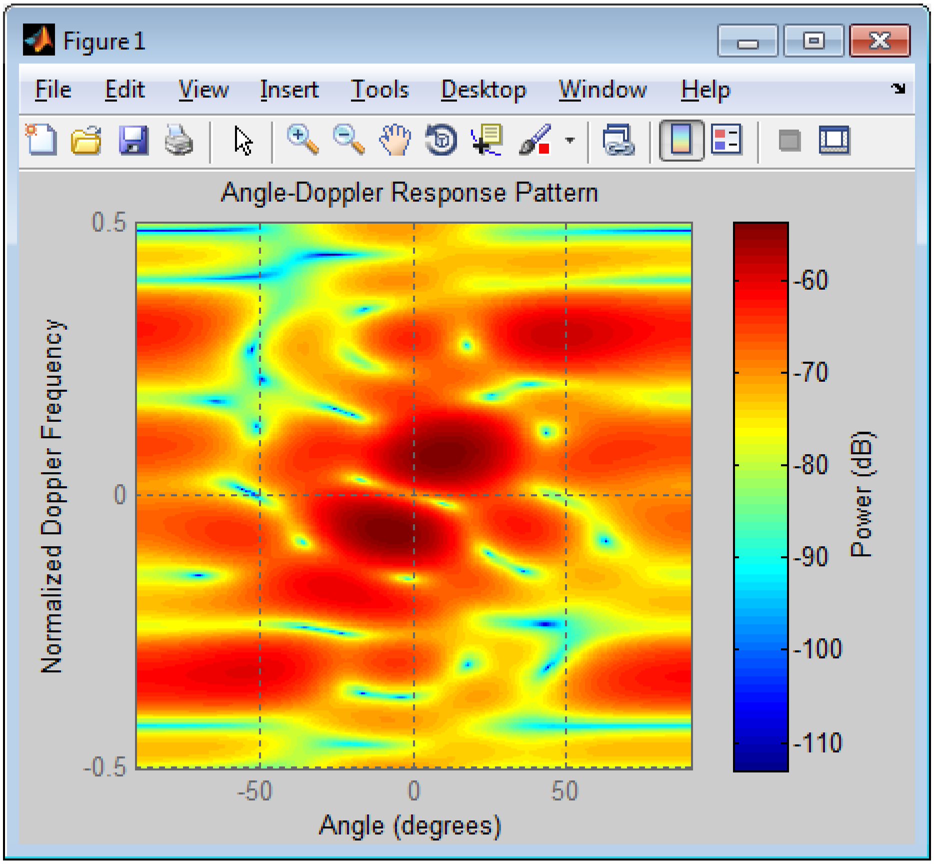
```
    'TransmitERP',tpower,'PlatformHeight',height,...
    'PlatformSpeed',speed,'PlatformDirection',direction,...
    'BroadsideDepressionAngle',depang,'MaximumRange',Rmax,...
    'AzimuthCoverage',Azcov,'SeedSource','Property',...
    'Seed',40547);
```

Simulate the clutter return for 10 pulses.

```
Nsamp = fs/prf; Npulse = 10;
csig = zeros(Nsamp,Nele,Npulse);
for m = 1:Npulse
    csig(:,:,m) = step(hclutter);
end
```

Plot the angle-Doppler response of the clutter at the 20th range bin.

```
hresp = phased.AngleDopplerResponse('SensorArray',ha,...
    'OperatingFrequency',fc,'PropagationSpeed',c,'PRF',prf);
plotResponse(hresp,shiftdim(csig(20,:,:)),...
    'NormalizeDoppler',true);
```



Clutter Simulation Using Known Transmit Signal

Simulate the clutter return from terrain with a gamma value of 0 dB. The `step` syntax includes the transmit signal of the radar system as an input argument. In this case, you do not record the effective transmitted power of the signal in a property.

Set up the characteristics of the radar system. This system has a 4-element uniform linear array (ULA). The sample rate is 1 MHz, and the PRF is 10 kHz. The propagation speed is 300,000 km/s, and the operating frequency is 300 MHz. The radar platform is flying 1 km above the ground with a path parallel to the ground along the array axis. The platform speed is 2000 m/s. The mainlobe has a depression angle of 30 degrees.

```
Nele = 4;
c = 3e8; fc = 3e8; lambda = c/fc;
ha = phased.ULA('NumElements',Nele,'ElementSpacing',lambda/2);

fs = 1e6; prf = 10e3;
height = 1000; direction = [90; 0];
speed = 2000; depang = 30;
```

Create the clutter simulation object and configure it to take a transmit signal as an input argument to `step`. The configuration assumes the earth is flat. The maximum clutter range of interest is 5 km, and the maximum azimuth coverage is +/– 60 degrees.

```
Rmax = 5000; Azcov = 120;
tergamma = 0;
hclutter = phased.ConstantGammaClutter('Sensor',ha,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'PRF',prf,...
    'SampleRate',fs,'Gamma',tergamma,'EarthModel','Flat',...
    'TransmitSignalInputPort',true,'PlatformHeight',height,...
    'PlatformSpeed',speed,'PlatformDirection',direction,...
    'BroadsideDepressionAngle',depang,'MaximumRange',Rmax,...
    'AzimuthCoverage',Azcov,'SeedSource','Property',...
    'Seed',40547);
```

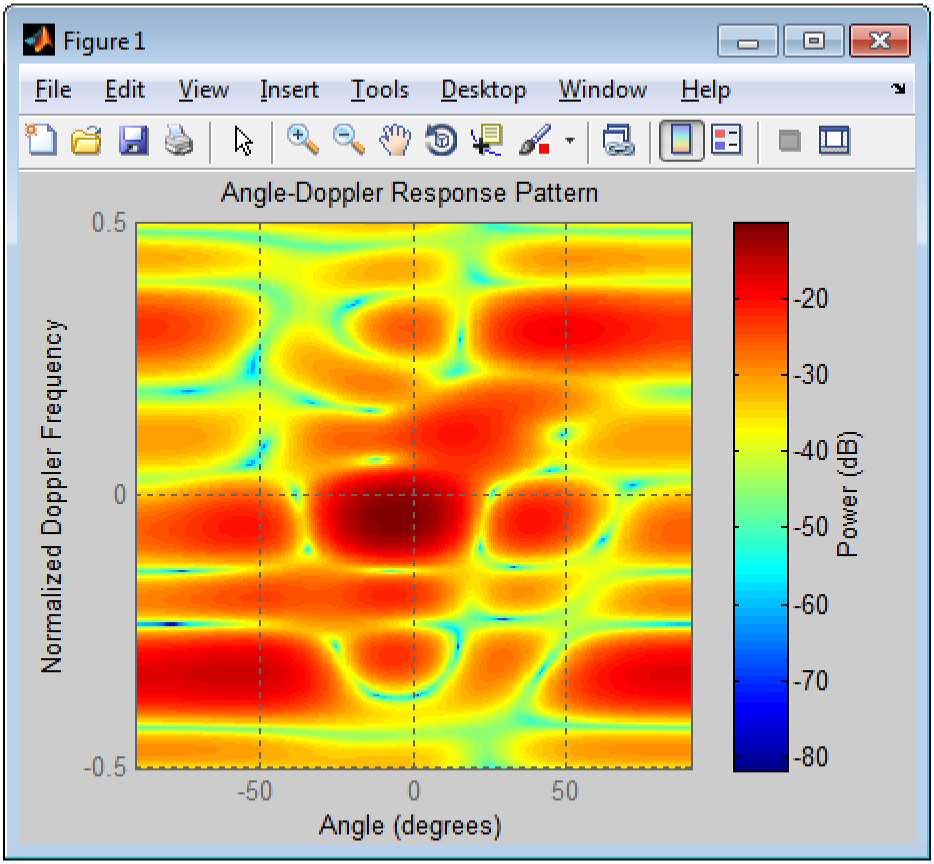
Simulate the clutter return for 10 pulses. At each step, pass the transmit signal as an input argument. The software automatically computes the effective transmitted power of the signal. The transmit signal is a rectangular waveform with a pulse width of 2 μs.

```
tpower = 5000;
pw = 2e-6;
X = tpower*ones(floor(pw*fs),1);
Nsamp = fs/prf; Npulse = 10;
```

```
csig = zeros(Nsamp,Nele,Npulse);
for m = 1:Npulse
    csig(:,:,m) = step(hclutter,X);
end
```

Plot the angle-Doppler response of the clutter at the 20th range bin.

```
hresp = phased.AngleDopplerResponse('SensorArray',ha,...
    'OperatingFrequency',fc,'PropagationSpeed',c,'PRF',prf);
plotResponse(hresp,shiftdim(csig(20,:,:)),...
    'NormalizeDoppler',true);
```



Extended Capabilities

Parallel Computing

You can use this System object to perform Monte Carlo simulations with Parallel Computing Toolbox constructs, such as `parfor`. In this situation, set the `SeedSource` property to `'Auto'` to ensure correct, automatic handling of random number streams on the workers.

Do not use this System object in a parallel construct whose iterations represent data from consecutive pulses. Because such iterations are not independent of each other, they must run sequentially. For more information about parallel computing constructs, see “Deciding When to Use parfor” or “Programming Considerations”.

To perform computations on a GPU instead of a CPU, use `phased.gpu.ConstantGammaClutter` instead of `phased.ConstantGammaClutter`.

References

[1] Barton, David. “Land Clutter Models for Radar Design and Analysis,” *Proceedings of the IEEE*. Vol. 73, Number 2, February, 1985, pp. 198–204.

[2] Long, Maurice W. *Radar Reflectivity of Land and Sea*, 3rd Ed. Boston: Artech House, 2001.

[3] Nathanson, Fred E., J. Patrick Reilly, and Marvin N. Cohen. *Radar Design Principles*, 2nd Ed. Mendham, NJ: SciTech Publishing, 1999.

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

See Also

`phased.BarrageJammer` | `phased.gpu.ConstantGammaClutter` | `surfacegamma` | `uv2azel` | `phitheta2azel`

Related Examples

- Ground Clutter Mitigation with Moving Target Indication (MTI) Radar
- “Example: DPCA Pulse Canceller for Clutter Rejection”

Concepts

- “Clutter Modeling”

Purpose Create constant gamma clutter simulation object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.ConstantGammaClutter.getNumInputs

Purpose Number of expected inputs to step method

Syntax

```
N = getNumInputs(H)
```

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ConstantGammaClutter.getNumOutputs

Purpose Number of outputs from step method

Syntax

```
N = getNumOutputs(H)
```

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.ConstantGammaClutter.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ConstantGammaClutter` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.ConstantGammaClutter.reset

Purpose Reset random numbers and time count for clutter simulation

Syntax `reset(H)`

Description `reset(H)` resets the states of the `ConstantGammaClutter` object, `H`. This method resets the random number generator state if the `SeedSource` property is set to `'Property'`. This method resets the elapsed coherence time. Also, if the `PRF` property is a vector, the next call to `step` uses the first PRF value in the vector.

phased.ConstantGammaClutter.step

Purpose

Simulate clutter using constant gamma model

Syntax

```
Y = step(H)
Y = step(H,X)
Y = step(H,STEERANGLE)
Y = step(H,X,STEERANGLE)
```

Description

`Y = step(H)` computes the collected clutter return at each sensor. This syntax is available when you set the `TransmitSignalInputPort` property to `false`.

`Y = step(H,X)` specifies the transmit signal in `X`. *Transmit signal* refers to the output of the transmitter while it is on during a given pulse. This syntax is available when you set the `TransmitSignalInputPort` property to `true`.

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

Input Arguments

H

Constant gamma clutter object.

X

Transmit signal, specified as a column vector.

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.

Output Arguments

Y

Collected clutter return at each sensor. `Y` has dimensions N-by-M matrix. M is the number of subarrays in the radar system if `H.Sensor` contains subarrays, or the number of sensors, otherwise. When you set the `OutputFormat` property to `'Samples'`, N is specified in the `NumSamples` property. When you set the `OutputFormat` property to `'Pulses'`, N is the total number of samples in the next *L* pulses. In this case, *L* is specified in the `NumPulses` property.

Tips

The clutter simulation that `ConstantGammaClutter` provides is based on these assumptions:

- The radar system is monostatic.
- The propagation is in free space.
- The terrain is homogeneous.
- The clutter patch is stationary during the coherence time. *Coherence time* indicates how frequently the software changes the set of random numbers in the clutter simulation.
- The signal is narrowband. Thus, the spatial response can be approximated by a phase shift. Similarly, the Doppler shift can be approximated by a phase shift.
- The radar system maintains a constant height during simulation.
- The radar system maintains a constant speed during simulation.

Examples

Clutter Simulation of System with Known Power

Simulate the clutter return from terrain with a gamma value of 0 dB. The effective transmitted power of the radar system is 5 kw.

Set up the characteristics of the radar system. This system has a 4-element uniform linear array (ULA). The sample rate is 1 MHz, and the PRF is 10 kHz. The propagation speed is 300,000 km/s, and the operating frequency is 300 MHz. The radar platform is flying 1 km above the ground with a path parallel to the ground along the array axis. The platform speed is 2000 m/s. The mainlobe has a depression angle of 30 degrees.

```
Nele = 4;
c = 3e8; fc = 3e8; lambda = c/fc;
ha = phased.ULA('NumElements',Nele,'ElementSpacing',lambda/2);

fs = 1e6; prf = 10e3;
height = 1000; direction = [90; 0];
speed = 2000; depang = 30;
```

Create the clutter simulation object. The configuration assumes the earth is flat. The maximum clutter range of interest is 5 km, and the maximum azimuth coverage is +/– 60 degrees.

```
Rmax = 5000; Azcov = 120;
tergamma = 0; tpower = 5000;
hclutter = phased.ConstantGammaClutter('Sensor',ha,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'PRF',prf,...
    'SampleRate',fs,'Gamma',tergamma,'EarthModel','Flat',...
    'TransmitERP',tpower,'PlatformHeight',height,...
    'PlatformSpeed',speed,'PlatformDirection',direction,...
    'BroadsideDepressionAngle',depang,'MaximumRange',Rmax,...
    'AzimuthCoverage',Azcov,'SeedSource','Property',...
    'Seed',40547);
```

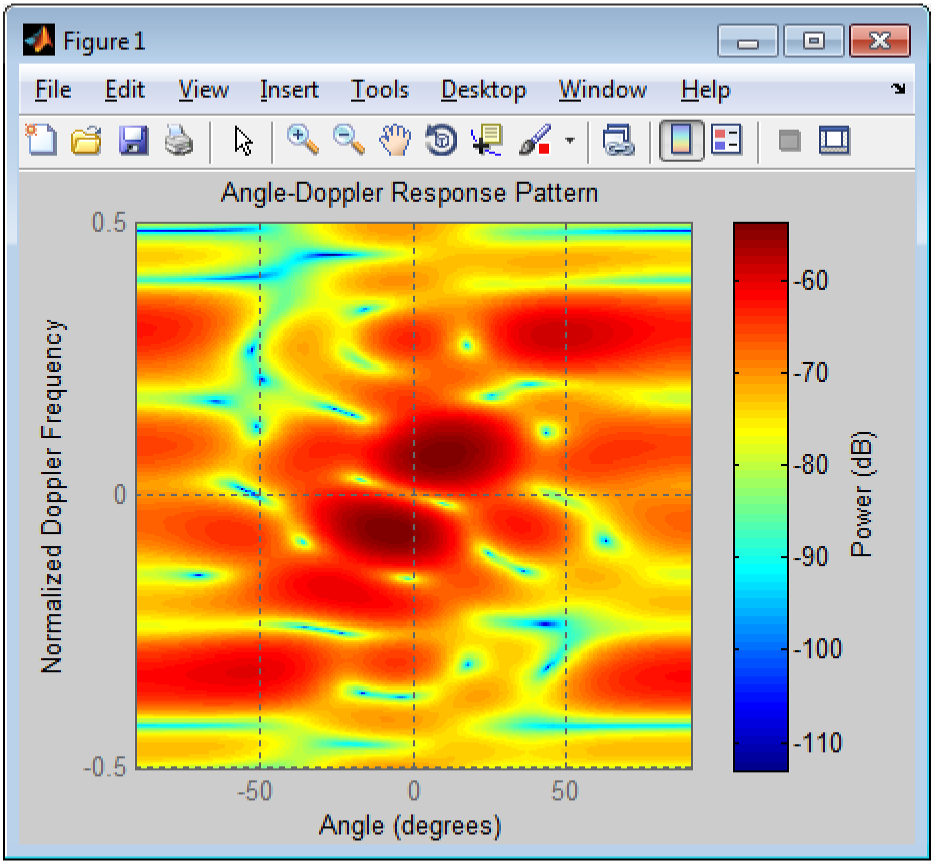
Simulate the clutter return for 10 pulses.

```
Nsamp = fs/prf; Npulse = 10;
```

```
csig = zeros(Nsamp,Nele,Npulse);
for m = 1:Npulse
    csig(:,:,m) = step(hclutter);
end
```

Plot the angle-Doppler response of the clutter at the 20th range bin.

```
hresp = phased.AngleDopplerResponse('SensorArray',ha,...
    'OperatingFrequency',fc,'PropagationSpeed',c,'PRF',prf);
plotResponse(hresp,shiftdim(csig(20,:,:)),...
    'NormalizeDoppler',true);
```



Clutter Simulation Using Known Transmit Signal

Simulate the clutter return from terrain with a gamma value of 0 dB. The `step` syntax includes the transmit signal of the radar system as an input argument. In this case, you do not record the effective transmitted power of the signal in a property.

Set up the characteristics of the radar system. This system has a 4-element uniform linear array (ULA). The sample rate is 1 MHz, and the PRF is 10 kHz. The propagation speed is 300,000 km/s, and the operating frequency is 300 MHz. The radar platform is flying 1 km above the ground with a path parallel to the ground along the array axis. The platform speed is 2000 m/s. The mainlobe has a depression angle of 30 degrees.

```
Nele = 4;
c = 3e8; fc = 3e8; lambda = c/fc;
ha = phased.ULA('NumElements',Nele,'ElementSpacing',lambda/2);

fs = 1e6; prf = 10e3;
height = 1000; direction = [90; 0];
speed = 2000; depang = 30;
```

Create the clutter simulation object and configure it to take a transmit signal as an input argument to `step`. The configuration assumes the earth is flat. The maximum clutter range of interest is 5 km, and the maximum azimuth coverage is +/– 60 degrees.

```
Rmax = 5000; Azcov = 120;
tergamma = 0;
hclutter = phased.ConstantGammaClutter('Sensor',ha,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'PRF',prf,...
    'SampleRate',fs,'Gamma',tergamma,'EarthModel','Flat',...
    'TransmitSignalInputPort',true,'PlatformHeight',height,...
    'PlatformSpeed',speed,'PlatformDirection',direction,...
    'BroadsideDepressionAngle',depang,'MaximumRange',Rmax,...
    'AzimuthCoverage',Azcov,'SeedSource','Property',...
    'Seed',40547);
```

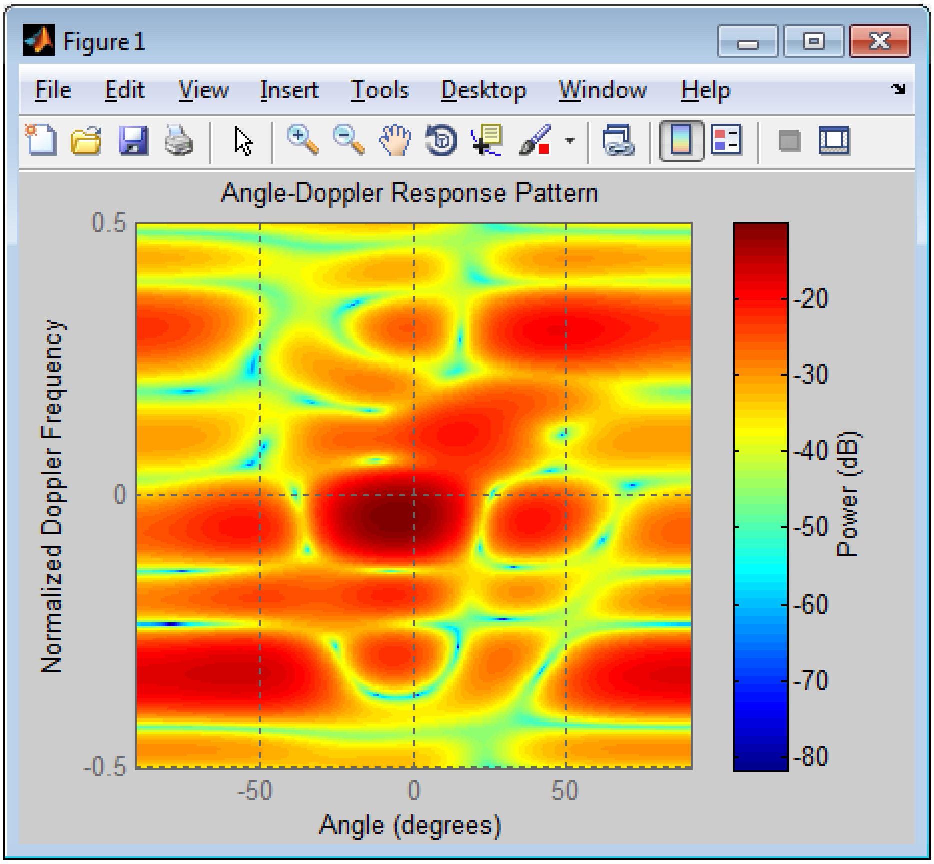
Simulate the clutter return for 10 pulses. At each step, pass the transmit signal as an input argument. The software automatically computes the effective transmitted power of the signal. The transmit signal is a rectangular waveform with a pulse width of 2 μs.

```
tpower = 5000;
pw = 2e-6;
X = tpower*ones(floor(pw*fs),1);
Nsamp = fs/prf; Npulse = 10;
```

```
csig = zeros(Nsamp,Nele,Npulse);
for m = 1:Npulse
    csig(:,:,m) = step(hclutter,X);
end
```

Plot the angle-Doppler response of the clutter at the 20th range bin.

```
hresp = phased.AngleDopplerResponse('SensorArray',ha,...
    'OperatingFrequency',fc,'PropagationSpeed',c,'PRF',prf);
plotResponse(hresp,shiftdim(csig(20,:,:)),...
    'NormalizeDoppler',true);
```



Related Examples

- Ground Clutter Mitigation with Moving Target Indication (MTI) Radar
- "Example: DPCA Pulse Canceller for Clutter Rejection"

Concepts

- "Clutter Modeling"

Purpose

Cosine antenna

Description

The `CosineAntennaElement` object models an antenna with a cosine response in both azimuth and elevation.

To compute the response of the antenna element for specified directions:

1. Define and set up your cosine antenna element. See “Construction” on page 3-225.

2. Call `step` to compute the antenna response according to the properties of `phased.CosineAntennaElement`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.CosineAntennaElement` creates a cosine antenna system object, `H`, that models an antenna element whose response is cosine raised to a specified power greater than or equal to one in both the azimuth and elevation directions.

`H = phased.CosineAntennaElement(Name,Value)` creates a cosine antenna 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

FrequencyRange

Operating frequency range

Specify the operating frequency range (in hertz) of the antenna element as a 1-by-2 row vector in the form of `[LowerBound HigherBound]`. The antenna element has no response outside the specified frequency range. The default value represents the UHF band.

Default: `[3e8 1e9]`

CosinePower

Exponent of cosine pattern

Specify the exponent of cosine pattern as a scalar or a 1-by-2 vector. All specified values must be real numbers greater than or equal to 1. When you set `CosinePower` to a scalar, both the azimuth direction cosine pattern and the elevation direction cosine pattern are raised to the specified value. When you set `CosinePower` 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.

Default: `[1.5 1.5]`

Methods

clone	Create cosine antenna object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of antenna
release	Allow property value and input characteristics changes
step	Output response of antenna element

Definitions

Cosine Response

The *cosine response*, or *cosine pattern*, is given by:

$$P(az,el) = \cos^{m}(az)\cos^{n}(el)$$

In this expression:

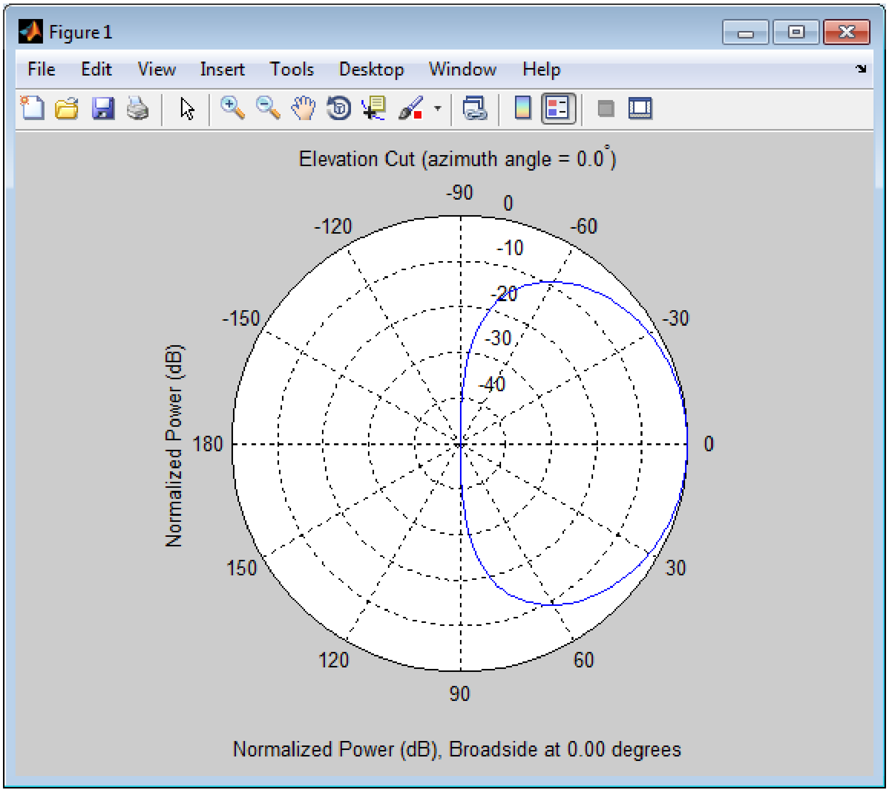
- *az* is the azimuth angle.
- *el* is the elevation angle.
- The exponents *m* and *n* are real numbers greater than or equal to 1.

The response is defined for azimuth and elevation angles between –90 and 90 degrees, inclusive. There is no response at the back of a cosine antenna. The cosine response pattern achieves a maximum value of 1 at 0 degrees azimuth and elevation. Raising the response pattern to powers greater than one concentrates the response in azimuth or elevation.

Examples

Construct a cosine pattern antenna and calculate its response at the boresight. Assume the antenna can work between 800 MHz and 1.2 GHz and the operating frequency is 1 GHz.

```
ha = phased.CosineAntennaElement('FrequencyRange',...
    [800e6 1.2e9]);
resp = step(ha,1e9,[0; 0]);
plotResponse(ha,1e9,'RespCut','El','Format','Polar');
```



See Also

`phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.URA` | `phased.ConformalArray` |

Purpose Create cosine antenna object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.CosineAntennaElement.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax

```
N = getNumOutputs(H)
```

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.CosineAntennaElement.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description `TF = isLocked(H)` returns the locked status, `TF` of the `CosineAntennaElement` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose

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 object.

FREQ

Operating frequency in hertz. If `FREQ` is a nonscalar row vector, the plot shows multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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

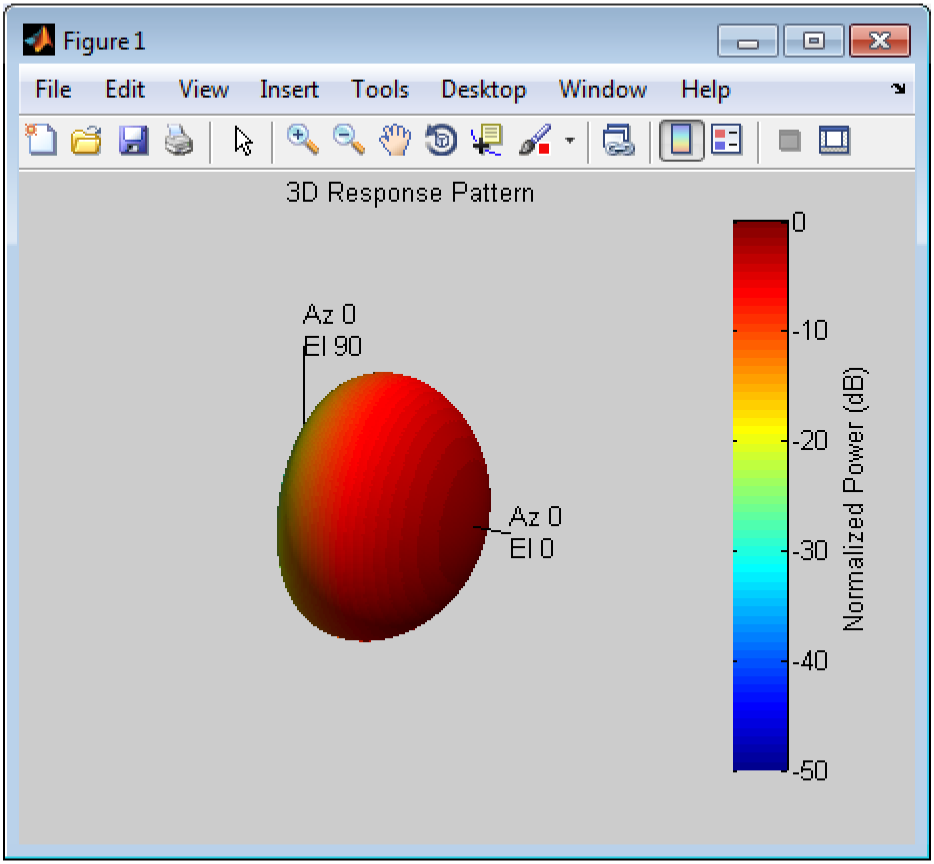
The unit of the plot. Valid values are `'db'`, `'mag'`, and `'pow'`.

Default: `'db'`

Examples

Construct a default cosine antenna. Assume the antenna operating frequency is 1 GHz. Plot the antenna's response as a polar plot in 3-D.

```
hcos = phased.CosineAntennaElement;
plotResponse(hcos,1e9,'Format','Polar','RespCut','3D');
```



See Also `uv2azel` | `azel2uv`

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.CosineAntennaElement.step

Purpose

Output response of antenna element

Syntax

`RESP = step(H,FREQ,ANG)`

Description

`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 `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

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. `RESP` is an M-by-L matrix. `RESP` contains the responses at the M angles specified in `ANG` and the L frequencies specified in `FREQ`.

Definitions

Cosine Response

The *cosine response*, or *cosine pattern*, is given by:

$$P(az,el) = \cos^m(az)\cos^n(el)$$

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 1.

The response is defined for azimuth and elevation angles between –90 and 90 degrees, inclusive. There is no response at the back of a cosine antenna. The cosine response pattern achieves a maximum value of 1 at 0 degrees azimuth and elevation. Raising the response pattern to powers greater than one concentrates the response in azimuth or elevation.

Examples

Construct a cosine antenna element. The cosine response is raised to a power of 1.5. The antenna frequency range is the IEEE® X band from 8 to 12 GHz. The antenna operates at 10 GHz. Obtain the antenna's response for an incident angle of 30 degrees azimuth and 5 degrees elevation.

```
hant = phased.CosineAntennaElement(...
    'FrequencyRange',[8e9 12e9],...
    'CosinePower',1.5);
% operating frequency
fc = 10e9;
% incident angle
```

```
ang = [30;5];
% use the step method to obtain the antenna's response
resp = step(hant,fc,ang);
```

See Also

`uv2azel` | `phitheta2azel`

Purpose

Custom antenna

Description

The `CustomAntennaElement` object models an antenna element with a custom response pattern.

To compute the response of the antenna element for specified directions:

1. Define and set up your custom antenna element. See "Construction" on page 3-241.
2. Call `step` to compute the antenna response according to the properties of `phased.CustomAntennaElement`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.CustomAntennaElement` creates a custom antenna system object, `H`. The object models an antenna element with a custom response pattern. The default custom antenna element has an isotropic response in the space.

`H = phased.CustomAntennaElement(Name,Value)` creates a custom antenna 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

FrequencyVector

Operating frequency vector

Specify the operating frequencies of the antenna element in hertz as a vector. The elements of the vector must be increasing. The antenna element has no response outside the frequency range specified by the minimum and maximum elements of the frequency vector.

Default: `[3e8 1e9]`

AzimuthAngles

Azimuth angles

Specify the azimuth angles (in degrees) as a vector of length P. These values are the azimuth angles where the custom pattern is evaluated. P must be greater than 2. The azimuth angles must lie between –180 and 180 degrees.

Default: `[-180:180]`

ElevationAngles

Elevation angles

Specify the elevation angles (in degrees) as a vector of length Q. These values are the elevation angles where the custom pattern is evaluated. Q must be greater than 2. The elevation angles must lie between –90 and 90 degrees.

Default: `[-90:90]`

FrequencyResponse

Frequency responses

Specify the frequency responses in decibels measured at the frequencies defined in `FrequencyVector` property as a row vector. The length of the vector must equal to the length of the frequency vector specified in the `FrequencyVector` property.

Default: `[0 0]`

RadiationPattern

Antenna radiation pattern

Specify the 3-D custom magnitude pattern in decibels as a Q-by-P matrix. Q is the number of elements in the `ElevationAngles` property and P is the number of elements in the `AzimuthAngles` 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 should cover azimuth angles in the range [–180, 180] degrees and elevation angles in the range [–90, 90] degrees.

If a particular value in the response pattern matrix is NaN, the processing considers the response to be zero at that point.

Default: A 181-by-361 matrix with all elements equal to 1

Methods

clone	Create custom antenna object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of antenna
release	Allow property value and input characteristics changes
step	Output response of antenna element

Examples

Response of Custom Antenna

Create a user-defined antenna with cosine pattern, and calculate that antenna's response at the boresight.

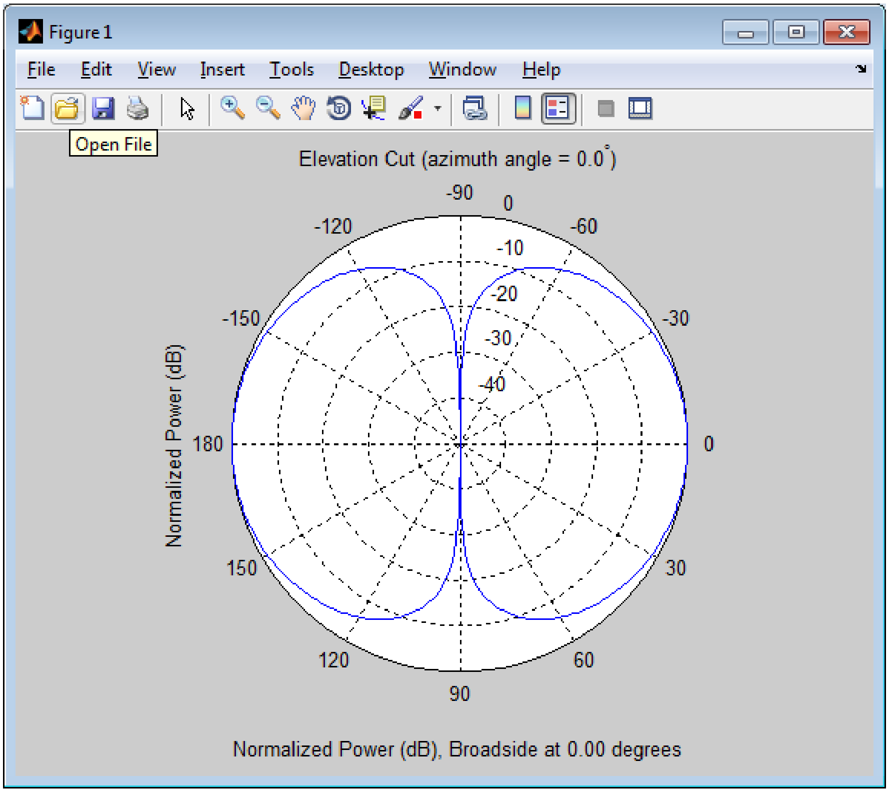
Create the antenna and calculate the response. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna works at 1 GHz.

```
ha = phased.CustomAntennaElement;
ha.AzimuthAngles = -180:180;
```

```
ha.ElevationAngles = -90:90;
ha.RadiationPattern = mag2db(repmat(cosd(ha.ElevationAngles)',...
    1,numel(ha.AzimuthAngles)));
resp = step(ha,1e9,[0; 0]);
```

Plot the response.

```
plotResponse(ha,1e9,'RespCut','El','Format','Polar');
```



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 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 that has this radiation pattern.

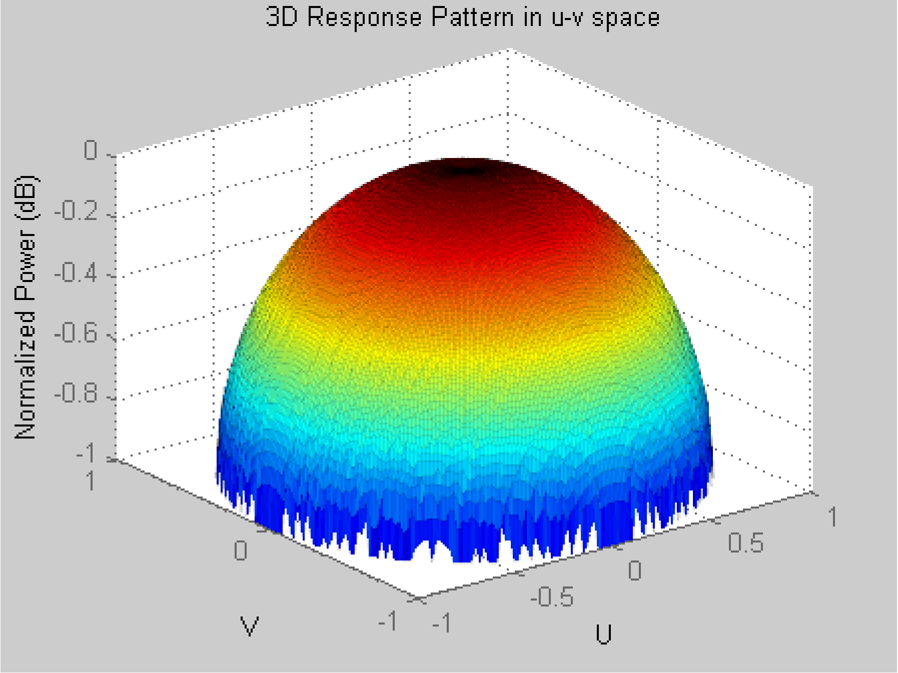
```
[pat_azel,az,el] = uv2azelpat(pat_uv,u,v);
ha = phased.CustomAntennaElement(...
    'AzimuthAngles',az,'ElevationAngles',el,...
    'RadiationPattern',pat_azel);
```

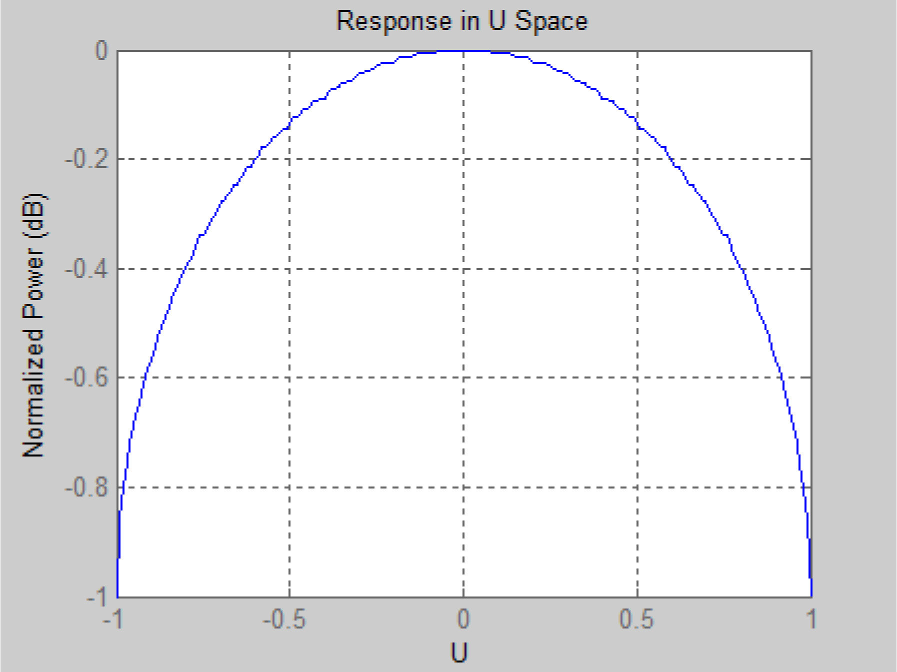
Calculate the response in the direction $u = 0.5$, $v = 0$. Assume the antenna operates at 1 GHz.

```
dir_uv = [0.5; 0];
dir_azel = uv2azel(dir_uv);
fc = 1e9;
resp = step(ha,fc,dir_azel);
```

Plot the response in *u*/*v* space as a 3-D plot and a *u* cut.

```
plotResponse(ha,fc,'Format','UV','RespCut','3D');
figure;
plotResponse(ha,fc,'Format','UV');
```


3D Response Pattern in u-v space
Normalized Power (dB)
0
-0.2
-0.4
-0.6
-0.8
-1
1
0
-1
V
-1
-0.5
0
0.5
1
U



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

`phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.URA` | `uv2azelpat` | `phitheta2azelpat` | `uv2azel` | `phitheta2azel`

phased.CustomAntennaElement.clone

Purpose Create custom antenna object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose

Number of expected inputs to step method

Syntax

`N = getNumInputs(H)`

Description

`N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.CustomAntennaElement.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.CustomAntennaElement.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `CustomAntennaElement` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.CustomAntennaElement.plotResponse

Purpose

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 object.

FREQ

Operating frequency in hertz. If `FREQ` is a nonscalar row vector, the plot shows multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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'`, and `'pow'`.

Default: `'db'`

Examples

Response of Custom Antenna

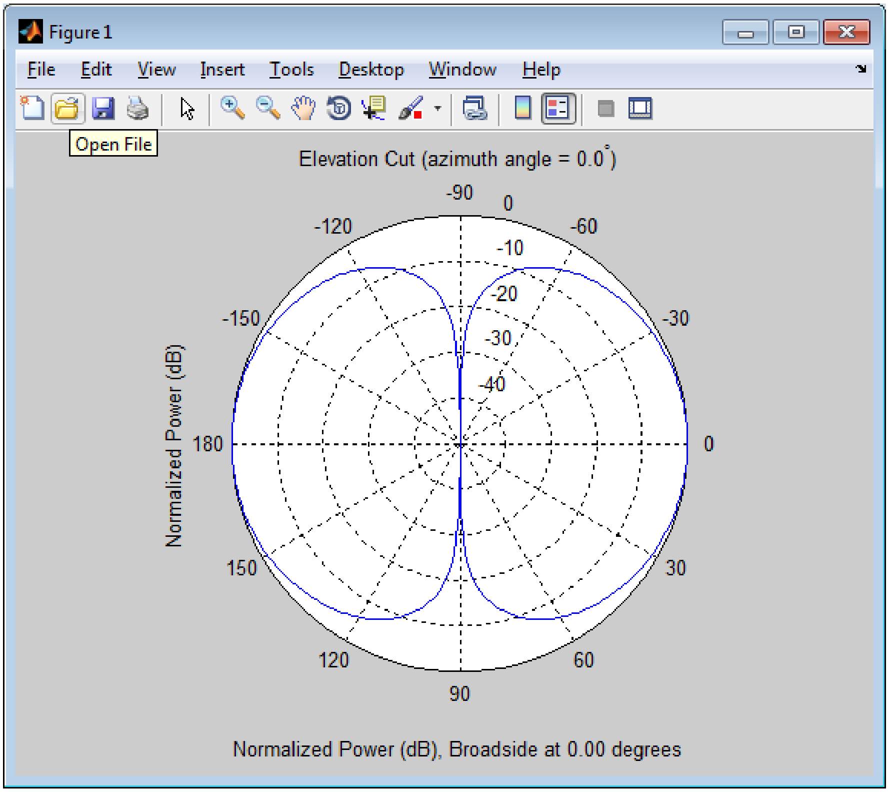
Create a user-defined antenna with cosine pattern, and plot that antenna's response.

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.

```
ha = phased.CustomAntennaElement;
ha.AzimuthAngles = -180:180;
ha.ElevationAngles = -90:90;
ha.RadiationPattern = mag2db(repmat(cosd(ha.ElevationAngles)',...
    1,numel(ha.AzimuthAngles)));
resp = step(ha,1e9,[0; 0]);
```

Plot the response.

```
plotResponse(ha,1e9,'RespCut','El','Format','Polar');
```



See Also

`uv2azel` | `azel2uv`

phased.CustomAntennaElement.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Output response of antenna element

Syntax

`RESP = step(H,FREQ,ANG)`

Description

`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 `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

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. `RESP` is an M-by-L matrix. `RESP` contains the responses at the M angles specified in `ANG` and the L frequencies specified in `FREQ`.

Examples

Construct a user defined antenna with an omnidirectional response in azimuth and a cosine pattern in elevation. The antenna operates at 1 GHz. Find the response of the antenna at the boresight.

```
ha = phased.CustomAntennaElement;
ha.AzimuthAngles = -180:180;
ha.ElevationAngles = -90:90;
ha.RadiationPattern = mag2db(repmat(cosd(ha.ElevationAngles)',...
    1,numel(ha.AzimuthAngles)));
resp = step(ha,1e9,[0; 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`

Purpose Custom microphone

Description The `CustomMicrophoneElement` object creates a custom microphone element.

To compute the response of the microphone element for specified directions:

1. Define and set up your custom microphone element. See "Construction" on page 3-259.
2. Call `step` to compute the response according to the properties of `phased.CustomMicrophoneElement`. The behavior of `step` is specific to each object in the toolbox.

Construction `H = phased.CustomMicrophoneElement` creates a custom microphone system object, `H`, that models a custom microphone element.

`H = phased.CustomMicrophoneElement(Name,Value)` creates a custom microphone 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

FrequencyVector

Operating frequency vector

Specify the frequencies in hertz where the frequency responses of element are measured as a vector. The elements of the vector must be increasing. The microphone element has no response outside the specified frequency range.

Default: `[20 20e3]`

FrequencyResponse

Frequency responses

Specify the frequency responses in decibels measured at the frequencies defined in the `FrequencyVector` property as a row vector. The length of the vector must equal the length of the frequency vector specified in the `FrequencyVector` property.

Default: `[0 0]`

PolarPatternFrequencies

Polar pattern measuring frequencies

Specify the measuring frequencies in hertz of the polar patterns as a row vector of length M. The measuring frequencies must be within the frequency range specified in the `FrequencyVector` property.

Default: `1e3`

PolarPatternAngles

Polar pattern measuring angles

Specify the measuring angles in degrees of the polar patterns as a row vector of length N. The angles are measured from the central pickup axis of the microphone, and must be between –180 and 180, inclusive.

Default: `[-180:180]`

PolarPattern

Polar pattern

Specify the polar patterns of the microphone element as an M-by-N 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 (in decibels) 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 0 degrees azimuth and 0 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.

Default: An omnidirectional pattern with 0 dB response everywhere

Methods

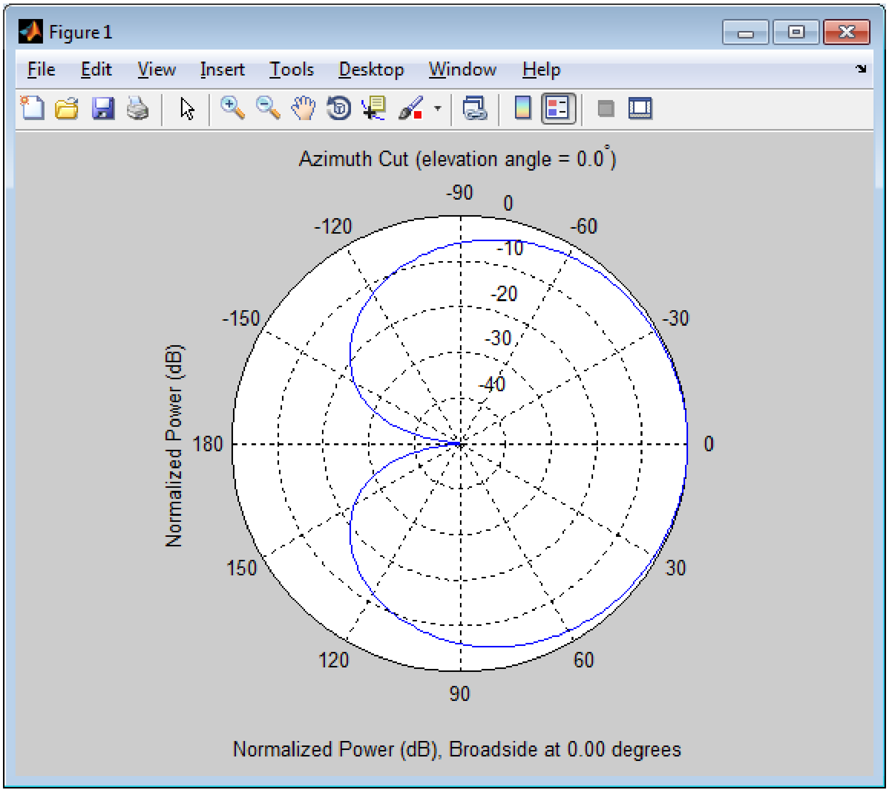
clone	Create omnidirectional microphone object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of microphone
release	Allow property value and input characteristics changes
step	Output response of microphone

Examples

Create a custom Cardioid microphone, and calculate that microphone's response at response at 500, 1500, and 2000 Hz in the directions `[0;0]` and `[40;50]`.

```
h = phased.CustomMicrophoneElement;
h.PolarPatternFrequencies = [500 1000];
h.PolarPattern = mag2db([...
    0.5+0.5*cosd(h.PolarPatternAngles);...
    0.6+0.4*cosd(h.PolarPatternAngles)]);
```

```
resp = step(h,[500 1500 2000],[0 0;40 50]');
plotResponse(h,500,'RespCut','Az','Format','Polar');
```



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

`phased.OmnidirectionalMicrophoneElement` | `phased.ULA` | `phased.URA` | `phased.ConformalArray` | `uv2azel` | `phitheta2azel`

phased.CustomMicrophoneElement.clone

Purpose Create omnidirectional microphone object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.CustomMicrophoneElement.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.CustomMicrophoneElement.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.CustomMicrophoneElement.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF` of the `CustomMicrophoneElement` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.CustomMicrophoneElement.plotResponse

Purpose

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 object.

FREQ

Operating frequency in hertz. If `FREQ` is a nonscalar row vector, the plot shows multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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'`, and `'pow'`.

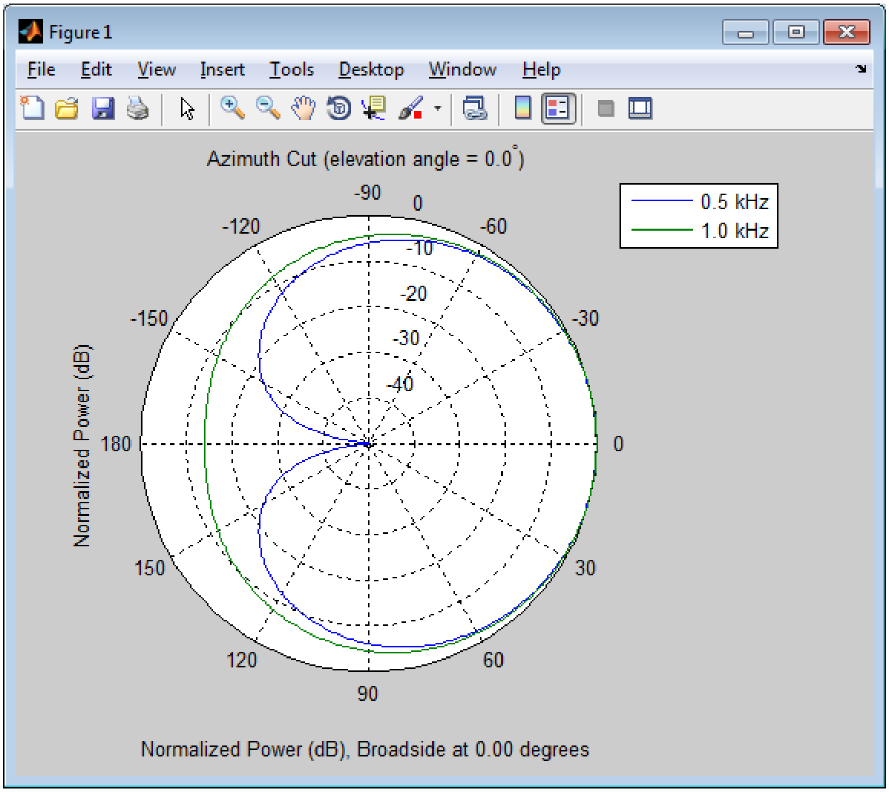
Default: `'db'`

Examples

Azimuth Response of Cardioid Microphone

Plot the azimuth responses of a custom cardioid microphone at operating frequencies of 500 Hz and 1 kHz.

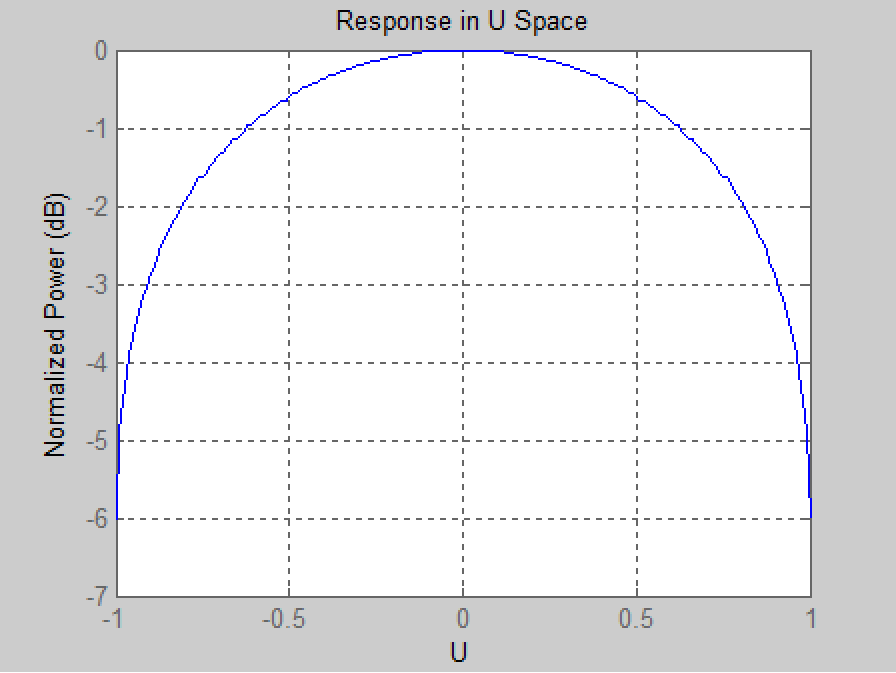
```
h = phased.CustomMicrophoneElement;
h.PolarPatternFrequencies = [500 1000];
h.PolarPattern = mag2db([...
    0.5+0.5*cosd(h.PolarPatternAngles);...
    0.6+0.4*cosd(h.PolarPatternAngles)]);
fc = 500;
plotResponse(h,[fc 2*fc],'RespCut','Az','Format','Polar');
```



Response of Cardioid Microphone in U/V Space

Plot the *u* cut of the response of a custom cardioid microphone in *u*/*v* space.

```
h = phased.CustomMicrophoneElement;
h.PolarPatternFrequencies = [500 1000];
h.PolarPattern = mag2db([...
    0.5+0.5*cosd(h.PolarPatternAngles);...
    0.6+0.4*cosd(h.PolarPatternAngles)]);
fc = 500;
plotResponse(h,fc,'Format','UV');
```



See Also `uv2azel` | `azel2uv`

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.CustomMicrophoneElement.step

Purpose

Output response of microphone

Syntax

`RESP = step(H,FREQ,ANG)`

Description

`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 `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

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

Construct a custom cardioid microphone with an operating frequency of 500 Hz. Find the microphone response in the directions of [0;0] and [40;50].

```
h = phased.CustomMicrophoneElement;
h.PolarPatternFrequencies = [500 1000];
h.PolarPattern = mag2db([...
    0.5+0.5*cosd(h.PolarPatternAngles);...
    0.6+0.4*cosd(h.PolarPatternAngles)]);
fc = 500; ang = [0 0;40 50]';
resp = step(h,fc,ang);
```

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`

Purpose

Displaced phase center array (DPCA) pulse canceller

Description

The `DPCACanceller` object implements a displaced phase center array pulse canceller.

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

1. Define and set up your DPCA pulse canceller. See "Construction" on page 3-276.
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.

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

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.

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`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency (PRF) of the received signal in hertz as a scalar.

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'`.

Default: `[0; 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'`.

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`

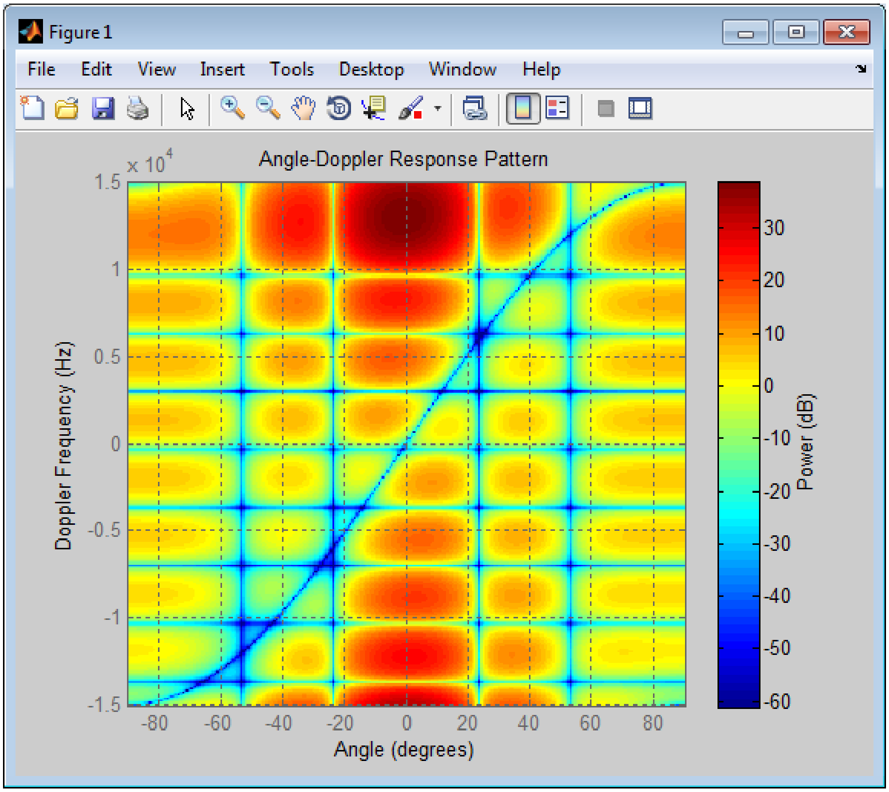
Methods

clone	Create DPCA object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform DPCA processing on input data

Examples

Process the data cube using a DPCA processor. The weights are calculated for the 71st cell of a collected data cube. The look direction is [0; 0] degrees and the Doppler is 12980 Hz.

```
load STAPExampleData;   % load data
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],12980);
Hresp = phased.AngleDopplerResponse(...
    'SensorArray',Hs.SensorArray,...
    'OperatingFrequency',Hs.OperatingFrequency,...
    'PRF',Hs.PRF,...
    'PropagationSpeed',Hs.PropagationSpeed);
plotResponse(Hresp,w);
```



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.

See Also

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

phased.DPCACanceller.clone

Purpose Create DPCA object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.DPCACanceller.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `DPCACanceller` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.DPCACanceller.release

Purpose

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform DPCA processing on input data

Syntax

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

Description

`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 mainlobe direction. This syntax is available when the `DirectionSource` property is `'Input port'` and the `DopplerSource` property is `'Property'`.

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

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

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

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).

CUTIDX

Range cell.

ANG

Receiving mainlobe 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.

Default: `Direction` property of `H`

DOP

Targeting Doppler frequency in hertz. `DOP` must be a scalar.

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 the data cube using a DPCA processor. The weights are calculated for the 71st cell of a collected data cube. The look direction is [0; 0] degrees and the Doppler is 12980 Hz.

```
load STAPExampleData;   % load data
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],12980);
```

See Also

`uv2azel` | `phitheta2azel`

phased.ElementDelay

Purpose

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 3-290.

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.

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.

Default: Speed of light

Methods

clone	Create element delay object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Calculate delay for elements

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 degrees azimuth and 20 degrees elevation.

```
ha = phased.ULA('NumElements',4);
hed = phased.ElementDelay('SensorArray',ha);
tau = step(hed,[30;20])
```

References

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

See Also

`phased.ArrayGain` **|** `phased.ArrayResponse` **|** `phased.SteeringVector` **|**

phased.ElementDelay.clone

Purpose Create element delay object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ElementDelay.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ElementDelay` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.ElementDelay.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Calculate delay for elements

Syntax

`TAU = step(H,ANG)`

Description

`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 `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

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.

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`.

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 degrees azimuth and 20 degrees elevation.

```
ha = phased.ULA('NumElements',4);
hed = phased.ElementDelay('SensorArray',ha);
tau = step(hed,[30;20])
```

See Also

`uv2azel` | `phitheta2azel`

Purpose

ESPRIT direction of arrival (DOA) estimator

Description

The `ESPRITEstimator` object computes a estimation of signal 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 3-299.
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.

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.

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`

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

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'`.

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 `(N-1)/2` where `N` is the number of elements of the array.

Default: `1`

VisibleRegion

Visible region

Specify the DOA search limits (in degrees) as a real 2-element row vector. The vector must be symmetric around broadside (0 degrees). This property applies when you set the `NumSignalsSource` property to `'Property'`.

Default: `[-90 90]`

Methods

clone	Create ESPRIT DOA estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform DOA estimation

Examples

Estimate the DOAs 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 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);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
hdoa = phased.ESPRITEstimator('SensorArray',ha,...
    'OperatingFrequency',fc);
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60])
```

References

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

See Also

`broadside2az`

phased.ESPRITEstimator.clone

Purpose Create ESPRIT DOA estimator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.ESPRITEstimator.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ESPRITEstimator.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `ESPRITEstimator` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.ESPRITEstimator.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform DOA estimation

Syntax

```
ANG = step(H,X)
```

Description

`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).

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.

Examples

Estimate the DOAs 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 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);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
hdoa = phased.ESPRITEstimator('SensorArray',ha,...
    'OperatingFrequency',fc);
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60])
```

Purpose

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 3-310.
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.

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, 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`

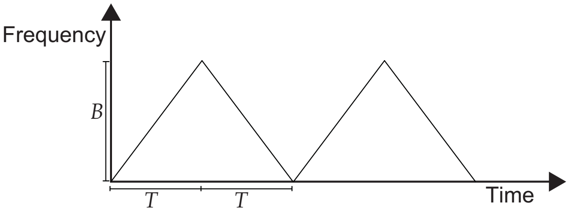
Methods

clone	Create FMCW waveform object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plot	Plot FMCW waveform
release	Allow property value and input characteristics changes
reset	Reset states of FMCW waveform object
step	Samples of FMCW waveform

Definitions

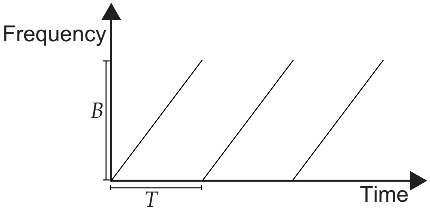
Triangle Sweep

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 $2T$.



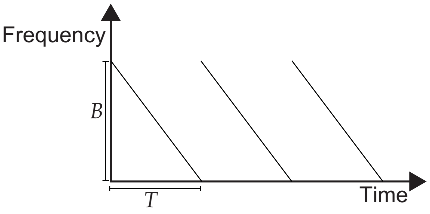
Upsweep

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.

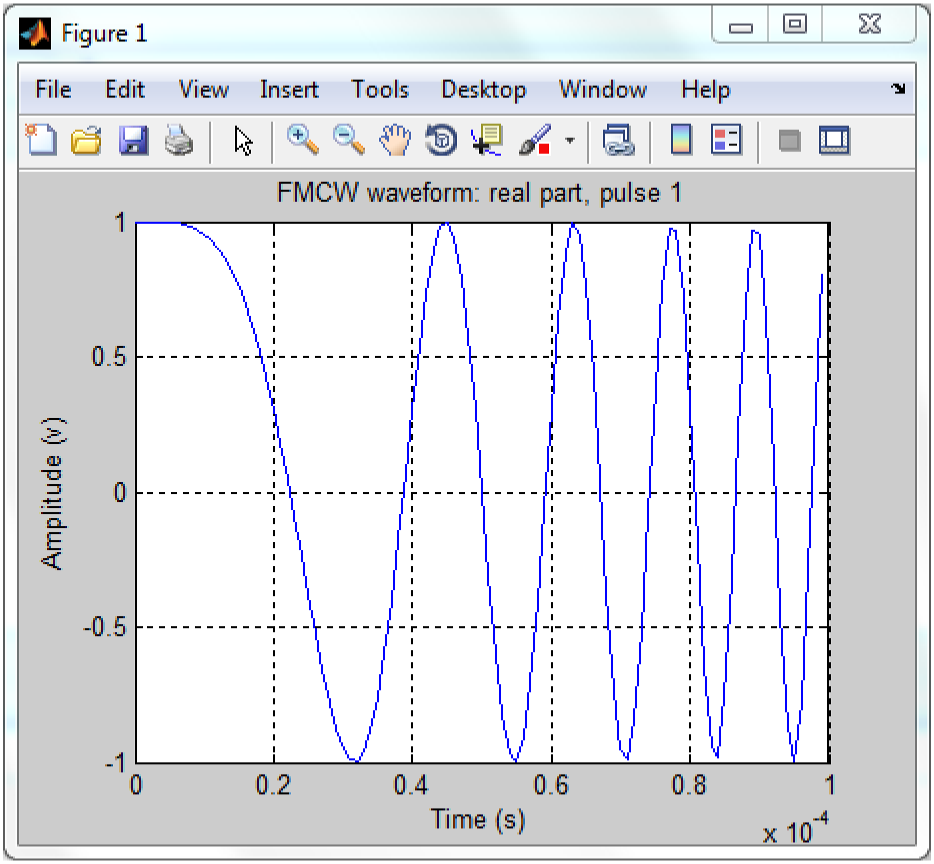


Examples

FMCW Waveform Plot

Create and plot an upsweep FMCW waveform.

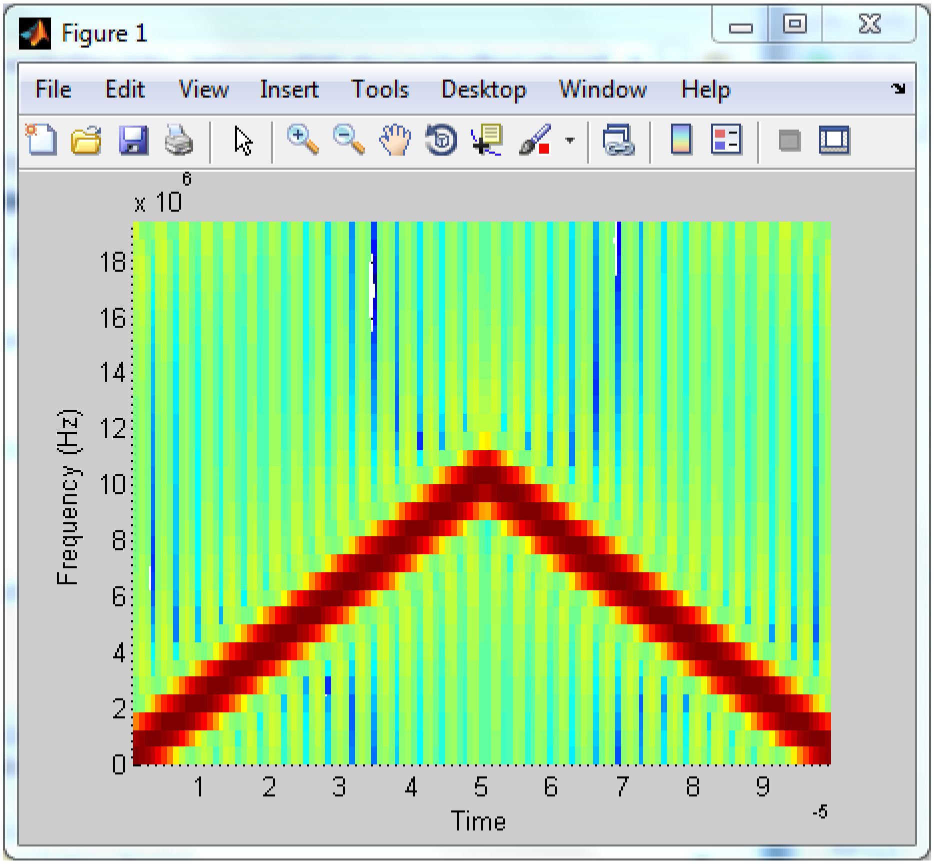
```
hw = phased.FMCWWaveform('SweepBandwidth',1e5,...
    'OutputFormat','Sweeps','NumSweeps',2);
plot(hw);
```



Spectrogram of Triangle Sweep FMCW Waveform

Generate samples of a triangle sweep FMCW Waveform. Then, examine the sweep using a spectrogram.

```
hw = phased.FMCWWaveform('SweepBandwidth',1e7,...
   'SampleRate',2e7,'SweepDirection','Triangle',...
   'NumSweeps',2);
x = step(hw);
spectrogram(x,32,16,32,hw.SampleRate,'yaxis');
```



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.

See Also

`range2time` | `time2range` | `range2bwphased.LinearFMWaveform` |

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

Purpose Create FMCW waveform object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.FMCWWaveform.getNumInputs

Purpose Number of expected inputs to step method

Syntax

```
N = getNumInputs(H)
```

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.FMCWWaveform.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `FMCWWaveform` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.FMCWWaveform.plot

Purpose

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

String that specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `Type` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Default: `'b'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

FMCW Waveform Plot

Create and plot an upsweep FMCW waveform.

```
hw = phased.FMCWWaveform('SweepBandwidth',1e5,...
    'OutputFormat','Sweeps','NumSweeps',2);
plot(hw);
```

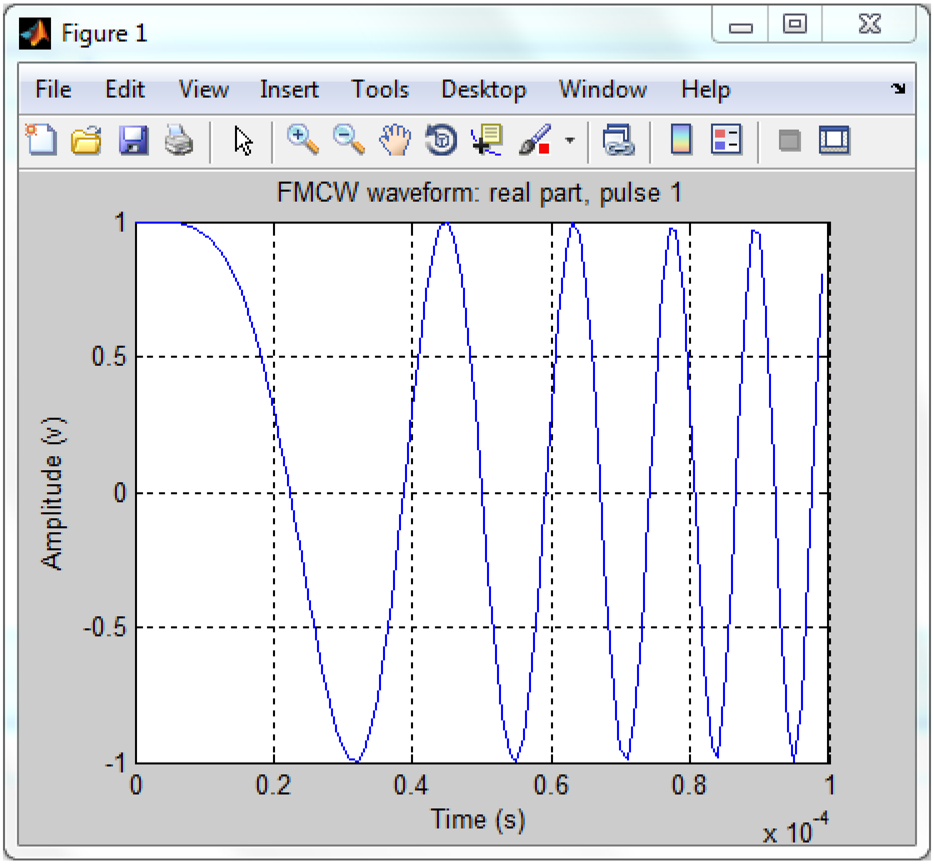


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
FMCW waveform: real part, pulse 1
1
0.5
0
-0.5
-1
Amplitude (v)
0
0.2
0.4
0.6
0.8
1
Time (s)
x 10^-4

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.FMCWWaveform.reset

Purpose 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.

Purpose

Samples of FMCW waveform

Syntax

`Y = step(H)`

Description

`Y = step(H)` returns samples of the FMCW waveform in a column vector, `Y`.

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

`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.

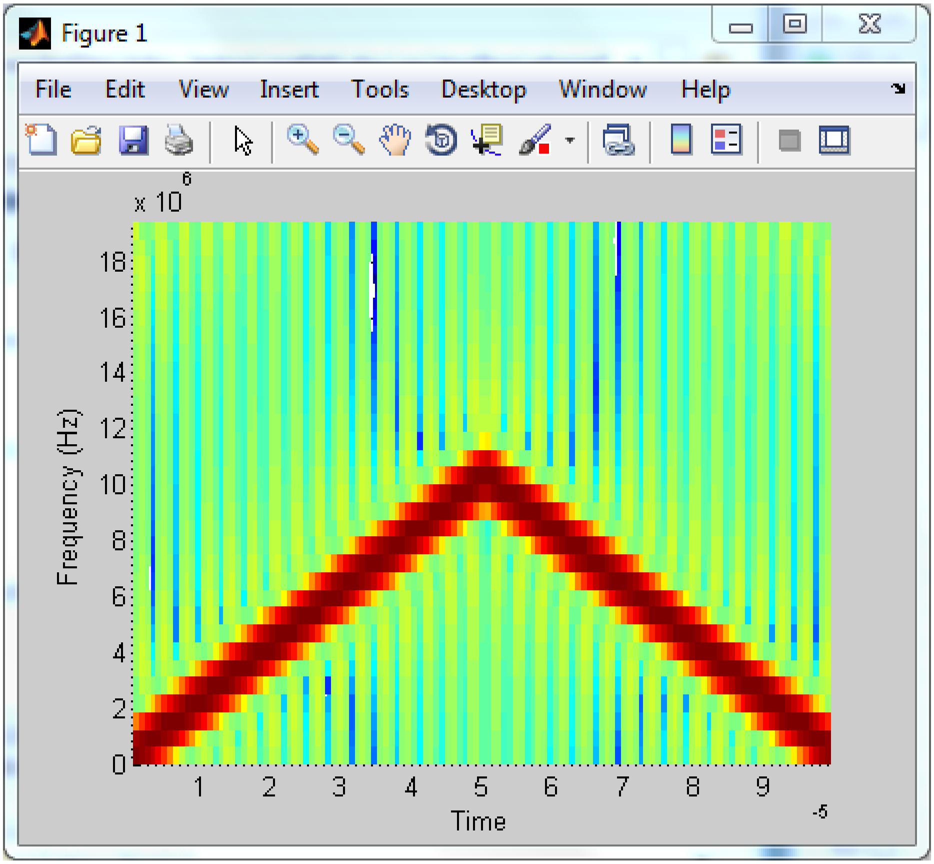
Examples

Spectrogram of Triangle Sweep FMCW Waveform

Generate samples of a triangle sweep FMCW Waveform. Then, examine the sweep using a spectrogram.

```
hw = phased.FMCWWaveform('SweepBandwidth',1e7,...
    'SampleRate',2e7,'SweepDirection','Triangle',...
```

```
    'NumSweeps',2);
x = step(hw);
spectrogram(x,32,16,32,hw.SampleRate,'yaxis');
```



Purpose

Free space environment

Description

The `FreeSpace` object models a free space environment.

To compute the propagated signal in free space:

1. Define and set up your free space environment. See "Construction" on page 3-329.
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.

Construction

`H = phased.FreeSpace` creates a free space environment System object, `H`. The object simulates narrowband signal propagation in free space, by applying range-dependent time delay, gain and phase shift to the input signal.

`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 the wave propagation speed (in meters per second) in free space as a scalar.

Default: Speed of light

OperatingFrequency

Signal carrier frequency

A scalar containing the carrier frequency in hertz of the narrowband signal. The default value of this property corresponds to 300 MHz.

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 (in hertz). The algorithm uses this value to determine the propagation delay in samples. The default value of this property corresponds to 1 MHz.

Default: `1e6`

Methods

clone	Create free space object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes

reset	Reset internal states of propagation channel
step	Propagate signal from one location to another

Examples

Signal Propagation from Stationary Radar to Stationary Target

Calculate the result of propagating a signal in a free space environment from a radar at (1000, 0, 0) to a target at (300, 200, 50). Assume both the radar and the target are stationary.

```
henv = phased.FreeSpace('SampleRate',8e3);
y = step(henv,ones(10,1),[1000; 0; 0],[300; 200; 50],...
   [0;0;0],[0;0;0]);
```

Signal Propagation from Moving Radar to Moving Target

Calculate the result of propagating a signal in a free space environment from a radar at (1000, 0, 0) to a target at (300, 200, 50). Assume the radar moves at 10 m/s in the direction of the *x*-axis, while the target moves at 15 m/s in the direction of the *y*-axis.

```
henv = phased.FreeSpace('SampleRate',8e3);
origin_pos = [1000; 0; 0];
dest_pos = [300; 200; 50];
origin_vel = [10; 0; 0];
dest_vel = [0; 15; 0];
y = step(henv,ones(10,1),origin_pos,dest_pos,...
   origin_vel,dest_vel);
```

Algorithms

When the origin and destination are stationary relative to each other, the output `Y` of `step` can be written as $Y(t)=x(t–tau)/L$. In this case, tau is the delay and L is the propagation loss. The delay tau is R/c, where R is the propagation distance and c is the propagation speed. The free space path loss is given by

$$L = \frac{(4\pi R)^2}{\lambda^2}$$

where λ is the signal wavelength.

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/λ for two-way propagation. In this case, v is the relative speed from the origin to the destination.

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

`fsplphased.RadarTarget` |

Purpose Create free space object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.FreeSpace.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.FreeSpace.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `FreeSpace` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.FreeSpace.reset

Purpose Reset internal states of propagation channel

Syntax `reset(H)`

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

Purpose

Propagate signal from one location to another

Syntax

```
Y = step(H,X,origin_pos,dest_pos,origin_vel,dest_vel)
```

Description

`Y = step(H,X,origin_pos,dest_pos,origin_vel,dest_vel)` returns the resulting signal, `Y`, when the narrowband signal `X` propagates in free space from `origin_pos` to `dest_pos`. The velocity of the signal origin is `origin_vel` and the velocity of the signal destination is `dest_vel`. Consider `FreeSpace` as a point-to-point propagation channel. For example, you can use it to model the propagation of a signal between a radar and a target.

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

H

Free space object.

X

Narrowband signal, specified as a column vector.

origin_pos

Starting location of signal, specified as a 3-by-1 column vector in the form `[x; y; z]` (in meters).

dest_pos

Ending location of signal, specified as a 3-by-1 column vector in the form `[x; y; z]` (in meters).

origin_vel

Velocity of signal origin, specified as a 3-by-1 column vector in the form `[Vx; Vy; Vz]` (in meters/second).

dest_vel

Velocity of the signal destination, specified as a 3-by-1 column vector in the form `[Vx; Vy; Vz]` (in meters/second).

Output Arguments

Y

Propagated signal, returned as a column vector. `Y` is the signal arriving at the propagation destination within the current time frame, which is the time occupied by the current input. If 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 result of propagating a signal in a free space environment from a radar at (1000, 0, 0) to a target at (300, 200, 50). Assume both the radar and the target are stationary.

```
henv = phased.FreeSpace('SampleRate',8e3);
y = step(henv,ones(10,1),[1000; 0; 0],[300; 200; 50],...
   [0;0;0],[0;0;0]);
```

Signal Propagation from Moving Radar to Moving Target

Calculate the result of propagating a signal in a free space environment from a radar at (1000, 0, 0) to a target at (300, 200, 50). Assume the radar moves at 10 m/s in the direction of the *x*-axis, while the target moves at 15 m/s in the direction of the *y*-axis.

```
henv = phased.FreeSpace('SampleRate',8e3);
origin_pos = [1000; 0; 0];
dest_pos = [300; 200; 50];
```

```
origin_vel = [10; 0; 0];
dest_vel = [0; 15; 0];
y = step(henv,ones(10,1),origin_pos,dest_pos,...
   origin_vel,dest_vel);
```

Algorithms

When the origin and destination are stationary relative to each other, the output `Y` of `step` can be written as $Y(t)=x(t–tau)/L$. In this case, tau is the delay and L is the propagation loss. The delay tau is R/c, where R is the propagation distance and c is the propagation speed. The free space path loss is given by

$$L = \frac{(4\pi R)^2}{\lambda^2}$$

where λ is the signal wavelength.

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. In this case, v is the relative speed from the origin to the destination.

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.

Purpose

Frost beamformer

Description

The `FrostBeamformer` object implements a Frost beamformer.

To compute the beamformed signal:

1. Define and set up your Frost beamformer. See "Construction" on page 3-342.
2. Call `step` to perform the beamforming operation according to the properties of `phased.FrostBeamformer`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.FrostBeamformer` creates a Frost beamformer System object, `H`. The object performs Frost beamforming on the received signal.

`H = phased.FrostBeamformer(Name,Value)` creates a Frost 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.

Default: Speed of light

SampleRate

Signal sampling rate

Specify the signal sampling rate (in hertz) as a positive scalar.

Default: `1e6`

FilterLength

FIR filter length

Specify the length of FIR filter behind each sensor element in the array as a positive integer.

Default: `2`

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.

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 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'`.

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

clone	Create Frost beamformer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform Frost beamforming

Examples

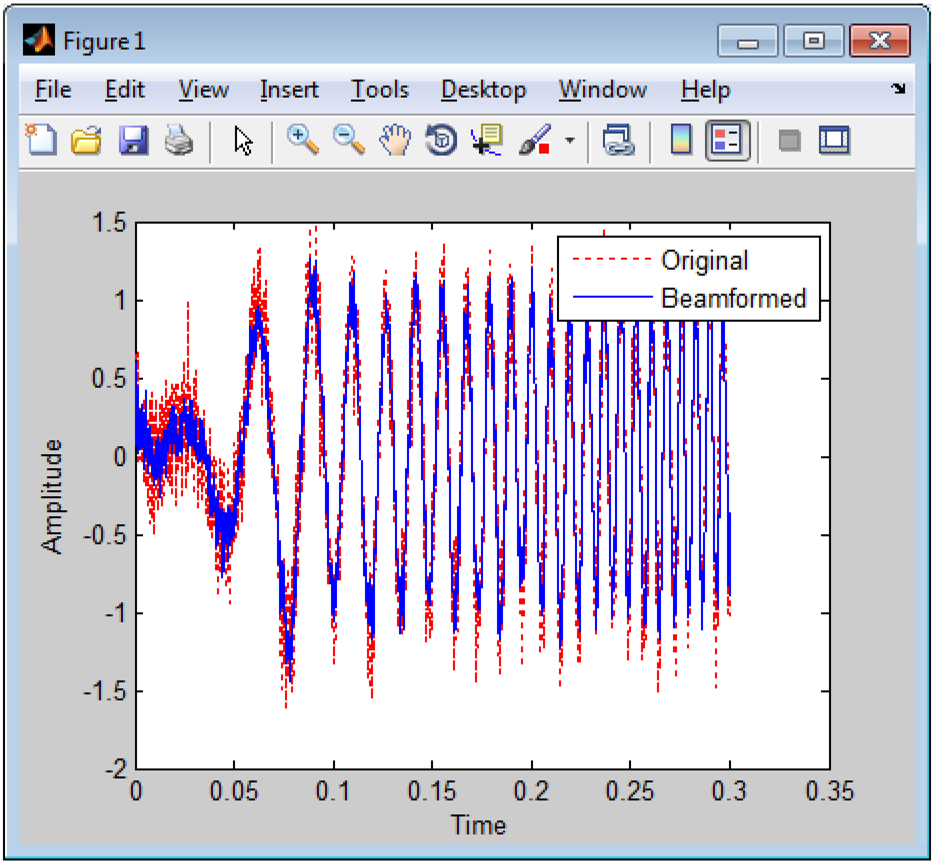
Apply a Frost beamformer to an 11-element array. The incident angle of the signal is –50 degrees in azimuth and 30 degrees in elevation.

```
% Signal simulation
ha = phased.ULA('NumElements',11,'ElementSpacing',0.04);
ha.Element.FrequencyRange = [20 20000];
fs = 8e3; t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340; % Wave propagation speed (m/s)
hc = phased.WidebandCollector('Sensor',ha,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',false);
incidentAngle = [-50; 30];
x = step(hc,x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x+noise;

% Beamforming
hbf = phased.FrostBeamformer('SensorArray',ha,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'Direction',incidentAngle,'FilterLength',5);
```

```
y = step(hbf,rx);

% Plot
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

`phased.PhaseShiftBeamformer` **|** `phased.SubbandPhaseShiftBeamformer` **|** `phased.TimeDelayBeamformer` **|** `phased.TimeDelayLCMVBeamformer` **|** `uv2azel` | `phitheta2azel`

phased.FrostBeamformer.clone

Purpose Create Frost beamformer object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.FrostBeamformer.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`N = getNumInputs(H)`

Description

`N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.FrostBeamformer.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.FrostBeamformer.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `FrostBeamformer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.FrostBeamformer.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

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

`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 `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

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.

XT

Training samples, specified as an M-by-N matrix. M and N are the same as the dimensions of `X`.

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 `getNumElements(H.SensorArray)*H.FilterLength`.

Examples

Apply a Frost beamformer to an 11-element array. The incident angle of the signal is –50 degrees in azimuth and 30 degrees in elevation.

```
% Signal simulation
ha = phased.ULA('NumElements',11,'ElementSpacing',0.04);
ha.Element.FrequencyRange = [20 20000];
fs = 8e3; t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340; % Wave propagation speed (m/s)
hc = phased.WidebandCollector('Sensor',ha,...
```

```
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',false);
incidentAngle = [-50; 30];
x = step(hc,x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x+noise;

% Beamforming
hbf = phased.FrostBeamformer('SensorArray',ha,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'Direction',incidentAngle,'FilterLength',5);
y = step(hbf,rx);
```

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.gpu.ConstantGammaClutter

Purpose

Constant gamma clutter simulation on GPU

Description

The `phased.gpu.ConstantGammaClutter` object simulates clutter, performing the computations on a GPU.

Note To use this object, you must install a Parallel Computing Toolbox license and have access to an appropriate GPU. For more about GPUs, see “GPU Computing” in the Parallel Computing Toolbox documentation.

To compute the clutter return:

1. Define and set up your clutter simulator. See “Construction” on page 3-357.
2. Call `step` to simulate the clutter return for your system according to the properties of `phased.gpu.ConstantGammaClutter`. The behavior of `step` is specific to each object in the toolbox.

The clutter simulation that `ConstantGammaClutter` provides is based on these assumptions:

- The radar system is monostatic.
- The propagation is in free space.
- The terrain is homogeneous.
- The clutter patch is stationary during the coherence time. *Coherence time* indicates how frequently the software changes the set of random numbers in the clutter simulation.
- The signal is narrowband. Thus, the spatial response can be approximated by a phase shift. Similarly, the Doppler shift can be approximated by a phase shift.
- The radar system maintains a constant height during simulation.

- The radar system maintains a constant speed during simulation.

Construction

`H = phased.gpu.ConstantGammaClutter` creates a constant gamma clutter simulation System object, `H`. This object simulates the clutter return of a monostatic radar system using the constant gamma model.

`H = phased.gpu.ConstantGammaClutter(Name,Value)` creates a constant gamma clutter simulation object, `H`, with additional options specified by one or more `Name,Value` pair arguments. `Name` is a property name, 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

Sensor

Handle of sensor

Specify the sensor as an isotropic antenna object or as an array object whose `Element` property value is an isotropic antenna element object. 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.

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`

SampleRate

Sample rate

Specify the sample rate, in hertz, as a positive scalar. The default value corresponds to 1 MHz.

Default: `1e6`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency in hertz as a positive scalar or a row vector. The default value of this property corresponds to 10 kHz. When `PRF` is a vector, it represents a staggered PRF. In this case, the output pulses use elements in the vector as their PRFs, one after another, in a cycle.

Default: `1e4`

Gamma

Terrain gamma value

Specify the γ value used in the constant γ clutter model, as a scalar in decibels. The γ value depends on both terrain type and the operating frequency.

Default: `0`

EarthModel

Earth model

Specify the earth model used in clutter simulation as one of | `'Flat'` | `'Curved'` |. When you set this property to `'Flat'`, the earth is assumed to be a flat plane. When you set this property to `'Curved'`, the earth is assumed to be a sphere.

Default: `'Flat'`

PlatformHeight

Radar platform height from surface

Specify the radar platform height (in meters) measured upward from the surface as a nonnegative scalar.

Default: `300`

PlatformSpeed

Radar platform speed

Specify the radar platform's speed as a nonnegative scalar in meters per second.

Default: `300`

PlatformDirection

Direction of radar platform motion

Specify the direction of radar platform motion as a 2-by-1 vector in the form [AzimuthAngle; ElevationAngle] in degrees. The default value of this property indicates that the platform moves perpendicular to the radar antenna array's broadside.

Both azimuth and elevation angle are measured in the local coordinate system of the radar antenna or antenna array. Azimuth angle must be between –180 and 180 degrees. Elevation angle must be between –90 and 90 degrees.

Default: `[90;0]`

BroadsideDepressionAngle

Depression angle of array broadside

Specify the depression angle in degrees of the broadside of the radar antenna array. This value is a scalar. The broadside is defined as zero degrees azimuth and zero degrees elevation. The depression angle is measured downward from horizontal.

Default: `0`

MaximumRange

Maximum range for clutter simulation

Specify the maximum range in meters for the clutter simulation as a positive scalar. The maximum range must be greater than the value specified in the `PlatformHeight` property.

Default: `5000`

AzimuthCoverage

Azimuth coverage for clutter simulation

Specify the azimuth coverage in degrees as a positive scalar. The clutter simulation covers a region having the specified azimuth span, symmetric to 0 degrees azimuth. Typically, all clutter patches have their azimuth centers within the region, but the `PatchAzimuthWidth` value can cause some patches to extend beyond the region.

Default: `60`

PatchAzimuthWidth

Azimuth span of each clutter patch

Specify the azimuth span of each clutter patch in degrees as a positive scalar.

Default: `1`

TransmitSignalInputPort

Add input to specify transmit signal

Set this property to `true` to add input to specify the transmit signal in the `step` syntax. Set this property to `false` omit the transmit signal in the `step` syntax. The `false` option is less computationally expensive; to use this option, you must also specify the `TransmitERP` property.

Default: `false`

TransmitERP

Effective transmitted power

Specify the transmitted effective radiated power (ERP) of the radar system in watts as a positive scalar. This property applies only when you set the `TransmitSignalInputPort` property to `false`.

Default: `5000`

CoherenceTime

Clutter coherence time

Specify the coherence time in seconds for the clutter simulation as a positive scalar. After the coherence time elapses, the `step` method updates the random numbers it uses for the clutter simulation at the next pulse. A value of `inf` means the random numbers are never updated.

Default: `inf`

OutputFormat

Output signal format

Specify the format of the output signal as one of `| 'Pulses' | 'Samples' |`. When you set the `OutputFormat` property to `'Pulses'`, the output of the `step` method is in the form of multiple pulses. In this case, the number of pulses is the value of the `NumPulses` property.

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. In staggered PRF applications, you might find the `'Samples'` option more convenient because the `step` output always has the same matrix size.

Default: `'Pulses'`

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`

NumSamples

Number of samples in output

Specify the number of samples in the output of the `step` method as a positive integer. Typically, you use the number of samples in one pulse. This property applies only when you set the `OutputFormat` property to `'Samples'`.

Default: `100`

SeedSource

Source of seed for random number generator

Specify how the object generates random numbers. Values of this property are:

`'Auto'`	Random numbers come from the global GPU random number stream. `'Auto'` is appropriate in a variety of situations. In particular, if you want to use a generator algorithm other than `mrg32k3a`, set `SeedSource` to `'Auto'`. Then, configure the global GPU random number stream to use the generator of your choice. You can configure the global GPU random number stream using `parallel.gpu.RandStream` and `parallel.gpu.RandStream.setGlobalStream`.
`'Property'`	Random numbers come from a private stream of random numbers. The stream uses the `mrg32k3a` generator algorithm, with a seed specified in the `Seed` property of this object. If you do not want clutter computations to affect the global GPU random number stream, set `SeedSource` to `'Property'`.

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

clone	Create GPU constant gamma clutter simulation object with same property values
getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset random numbers and time count for clutter simulation
step	Simulate clutter using constant gamma model

Examples

Clutter Simulation of System with Known Power

Simulate the clutter return from terrain with a gamma value of 0 dB. The effective transmitted power of the radar system is 5 kw.

Set up the characteristics of the radar system. This system has a 4-element uniform linear array (ULA). The sample rate is 1 MHz, and the PRF is 10 kHz. The propagation speed is 300,000 km/s, and the operating frequency is 300 MHz. The radar platform is flying 1 km above the ground with a path parallel to the ground along the array axis. The platform speed is 2000 m/s. The mainlobe has a depression angle of 30 degrees.

```
Nele = 4;
c = 3e8; fc = 3e8; lambda = c/fc;
ha = phased.ULA('NumElements',Nele,'ElementSpacing',lambda/2);

fs = 1e6; prf = 10e3;
height = 1000; direction = [90; 0];
speed = 2000; depang = 30;
```

Create the GPU clutter simulation object. The configuration assumes the earth is flat. The maximum clutter range of interest is 5 km, and the maximum azimuth coverage is +/– 60 degrees.

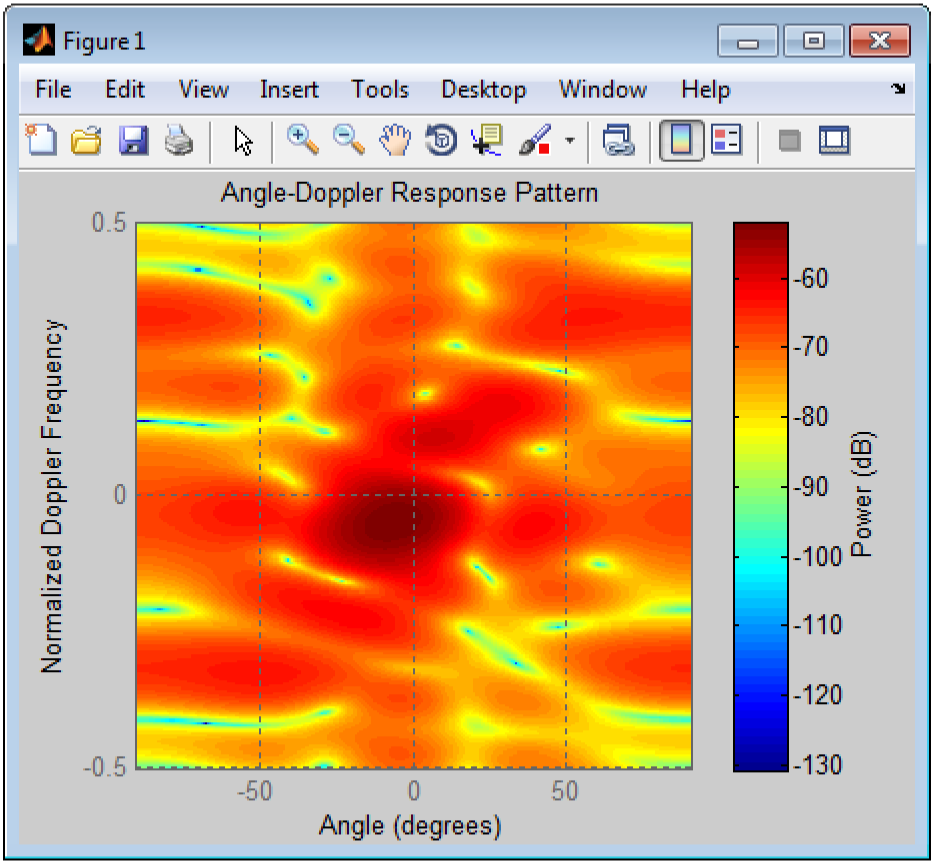
```
Rmax = 5000; Azcov = 120;
tergamma = 0; tpower = 5000;
hclutter = phased.gpu.ConstantGammaClutter('Sensor',ha,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'PRF',prf,...
    'SampleRate',fs,'Gamma',tergamma,'EarthModel','Flat',...
    'TransmitERP',tpower,'PlatformHeight',height,...
    'PlatformSpeed',speed,'PlatformDirection',direction,...
    'BroadsideDepressionAngle',depang,'MaximumRange',Rmax,...
    'AzimuthCoverage',Azcov);
```

Simulate the clutter return for 10 pulses.

```
Nsamp = fs/prf; Npulse = 10;
csig = zeros(Nsamp,Nele,Npulse);
for m = 1:Npulse
    csig(:,:,m) = step(hclutter);
end
```

Plot the angle-Doppler response of the clutter at the 20th range bin.

```
hresp = phased.AngleDopplerResponse('SensorArray',ha,...
    'OperatingFrequency',fc,'PropagationSpeed',c,'PRF',prf);
plotResponse(hresp,shiftdim(csig(20,:,:)),...
    'NormalizeDoppler',true);
```



The results do not exactly match those achieved by using `phased.ConstantGammaClutter` instead of `phased.gpu.ConstantGammaClutter`. This discrepancy occurs because of differences between CPU and GPU computations.

Clutter Simulation Using Known Transmit Signal

Simulate the clutter return from terrain with a gamma value of 0 dB. The `step` syntax includes the transmit signal of the radar system as an input argument. In this case, you do not record the effective transmitted power of the signal in a property.

Set up the characteristics of the radar system. This system has a 4-element uniform linear array (ULA). The sample rate is 1 MHz, and the PRF is 10 kHz. The propagation speed is 300,000 km/s, and the operating frequency is 300 MHz. The radar platform is flying 1 km above the ground with a path parallel to the ground along the array axis. The platform speed is 2000 m/s. The mainlobe has a depression angle of 30 degrees.

```
Nele = 4;
c = 3e8; fc = 3e8; lambda = c/fc;
ha = phased.ULA('NumElements',Nele,'ElementSpacing',lambda/2);

fs = 1e6; prf = 10e3;
height = 1000; direction = [90; 0];
speed = 2000; depang = 30;
```

Create the GPU clutter simulation object and configure it to take a transmit signal as an input argument to `step`. The configuration assumes the earth is flat. The maximum clutter range of interest is 5 km, and the maximum azimuth coverage is +/– 60 degrees.

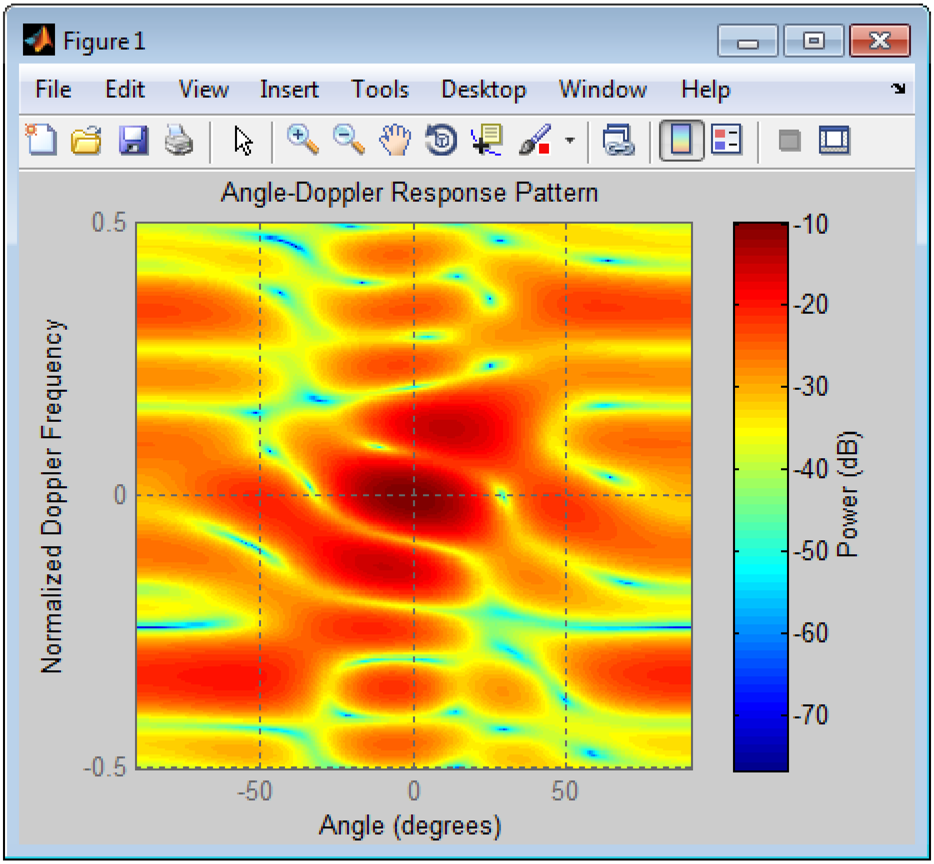
```
Rmax = 5000; Azcov = 120;
tergamma = 0;
hclutter = phased.gpu.ConstantGammaClutter('Sensor',ha,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'PRF',prf,...
    'SampleRate',fs,'Gamma',tergamma,'EarthModel','Flat',...
    'TransmitSignalInputPort',true,'PlatformHeight',height,...
    'PlatformSpeed',speed,'PlatformDirection',direction,...
    'BroadsideDepressionAngle',depang,'MaximumRange',Rmax,...
    'AzimuthCoverage',Azcov);
```

Simulate the clutter return for 10 pulses. At each step, pass the transmit signal as an input argument. The software automatically computes the effective transmitted power of the signal. The transmit signal is a rectangular waveform with a pulse width of 2 μs.

```
tpower = 5000;
pw = 2e-6;
X = tpower*ones(floor(pw*fs),1);
Nsamp = fs/prf; Npulse = 10;
csig = zeros(Nsamp,Nele,Npulse);
for m = 1:Npulse
    csig(:,:,m) = step(hclutter,X);
end
```

Plot the angle-Doppler response of the clutter at the 20th range bin.

```
hresp = phased.AngleDopplerResponse('SensorArray',ha,...
    'OperatingFrequency',fc,'PropagationSpeed',c,'PRF',prf);
plotResponse(hresp,shiftdim(csig(20,:,:)),...
    'NormalizeDoppler',true);
```



The results do not exactly match those achieved by using `phased.ConstantGammaClutter` instead of `phased.gpu.ConstantGammaClutter`. This discrepancy occurs because of differences between CPU and GPU computations.

Random Number Comparison Between GPU and CPU

In most cases, it does not matter that the GPU and CPU use different random numbers. Sometimes, you may need to reproduce the same stream on both GPU and CPU. In such cases, you can set up the two global streams so they produce identical random numbers. Both GPU and CPU support the combined multiple recursive generator (`mrg32k3a`) with the `NormalTransform` parameter set to `'Inversion'`.

Define a seed value to use for the GPU stream and the CPU stream.

```
seed = 7151;
```

Create a CPU random number stream that uses the combined multiple recursive generator and the chosen seed value. Then, use this stream as the global stream for random number generation on the CPU.

```
stream_cpu = RandStream('CombRecursive','Seed',seed,...
   'NormalTransform','Inversion');
RandStream.setGlobalStream(stream_cpu);
```

Create a GPU random number stream that uses the combined multiple recursive generator and the same seed value. Then, use this stream as the global stream for random number generation on the GPU.

```
stream_gpu = parallel.gpu.RandStream('CombRecursive','Seed',seed);
parallel.gpu.RandStream.setGlobalStream(stream_gpu);
```

Generate clutter on both the CPU and the GPU, using the global stream on each platform.

```
h_cpu = phased.ConstantGammaClutter('SeedSource','Auto');
h_gpu = phased.gpu.ConstantGammaClutter('SeedSource','Auto');

y_cpu = step(h_cpu);
y_gpu = step(h_gpu);
```

Check that the elementwise differences between the CPU and GPU results are negligible.

```
maxdiff = max(max(abs(y_cpu - y_gpu)))
eps
```

```
maxdiff =
```

```
   2.9092e-18


ans =

   2.2204e-16
```

References

[1] Barton, David. “Land Clutter Models for Radar Design and Analysis,” *Proceedings of the IEEE*. Vol. 73, Number 2, February, 1985, pp. 198–204.

[2] Long, Maurice W. *Radar Reflectivity of Land and Sea*, 3rd Ed. Boston: Artech House, 2001.

[3] Nathanson, Fred E., J. Patrick Reilly, and Marvin N. Cohen. *Radar Design Principles*, 2nd Ed. Mendham, NJ: SciTech Publishing, 1999.

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

See Also

`phased.BarrageJammer` **|** `phased.ConstantGammaClutter` **|** `surfacegamma` | `uv2azel` | `phitheta2azel`

Related Examples

- GPU Acceleration of Clutter Simulation
- Ground Clutter Mitigation with Moving Target Indication (MTI) Radar

Concepts

- “Clutter Modeling”
- “GPU Computing”

phased.gpu.ConstantGammaClutter.clone

Purpose Create GPU constant gamma clutter simulation object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.gpu.ConstantGammaClutter.getNumInputs

Purpose Number of expected inputs to step method

Syntax

```
N = getNumInputs(H)
```

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.gpu.ConstantGammaClutter.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ConstantGammaClutter` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.gpu.ConstantGammaClutter.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose Reset random numbers and time count for clutter simulation

Syntax `reset(H)`

Description `reset(H)` resets the states of the `ConstantGammaClutter` object, `H`. This method resets the random number generator state if the `SeedSource` property is set to `'Property'`. This method resets the elapsed coherence time. Also, if the `PRF` property is a vector, the next call to `step` uses the first PRF value in the vector.

phased.gpu.ConstantGammaClutter.step

Purpose

Simulate clutter using constant gamma model

Syntax

```
Y = step(H)
Y = step(H,X)
```

Description

`Y = step(H)` computes the collected clutter return at each sensor. This syntax is available when you set the `TransmitSignalInputPort` property to `false`.

`Y = step(H,X)` specifies the transmit signal in `X`. *Transmit signal* refers to the output of the transmitter while it is on during a given pulse. This syntax is available when you set the `TransmitSignalInputPort` property to `true`.

Input Arguments

H

Constant gamma clutter object.

X

Transmit signal, specified as a column vector of data type `double`. The System object handles data transfer between the CPU and GPU.

Output Arguments

Y

Collected clutter return at each sensor. The data types of `X` and `Y` are the same. `Y` has dimensions N-by-M matrix. M is the number of subarrays in the radar system if `H.Sensor` contains subarrays, or the number of sensors, otherwise. When you set the `OutputFormat` property to `'Samples'`, N is specified in the `NumSamples` property. When you set the `OutputFormat` property to `'Pulses'`, N is the total number of samples in the next L pulses. In this case, L is specified in the `NumPulses` property.

Tips

The clutter simulation that `ConstantGammaClutter` provides is based on these assumptions:

- The radar system is monostatic.

- The propagation is in free space.
- The terrain is homogeneous.
- The clutter patch is stationary during the coherence time. *Coherence time* indicates how frequently the software changes the set of random numbers in the clutter simulation.
- The signal is narrowband. Thus, the spatial response can be approximated by a phase shift. Similarly, the Doppler shift can be approximated by a phase shift.
- The radar system maintains a constant height during simulation.
- The radar system maintains a constant speed during simulation.

Examples

Clutter Simulation of System with Known Power

Simulate the clutter return from terrain with a gamma value of 0 dB. The effective transmitted power of the radar system is 5 kw.

Set up the characteristics of the radar system. This system has a 4-element uniform linear array (ULA). The sample rate is 1 MHz, and the PRF is 10 kHz. The propagation speed is 300,000 km/s, and the operating frequency is 300 MHz. The radar platform is flying 1 km above the ground with a path parallel to the ground along the array axis. The platform speed is 2000 m/s. The mainlobe has a depression angle of 30 degrees.

```
Nele = 4;
c = 3e8; fc = 3e8; lambda = c/fc;
ha = phased.ULA('NumElements',Nele,'ElementSpacing',lambda/2);

fs = 1e6; prf = 10e3;
height = 1000; direction = [90; 0];
speed = 2000; depang = 30;
```

Create the GPU clutter simulation object. The configuration assumes the earth is flat. The maximum clutter range of interest is 5 km, and the maximum azimuth coverage is +/– 60 degrees.

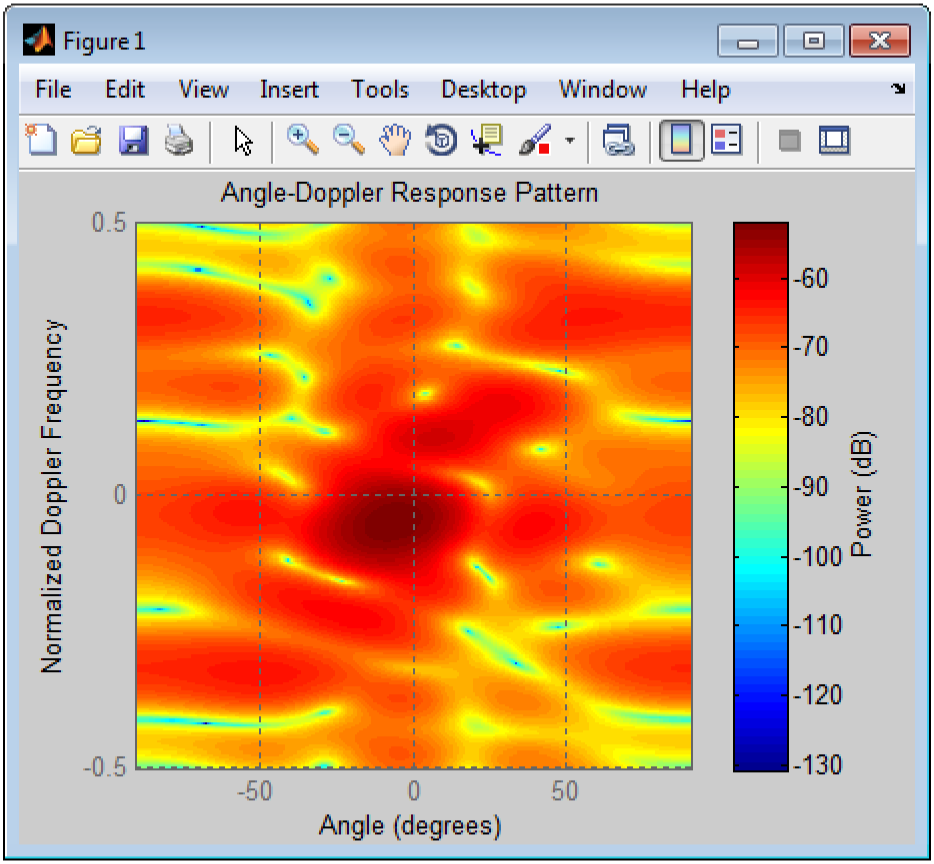
```
Rmax = 5000; Azcov = 120;
tergamma = 0; tpower = 5000;
hclutter = phased.gpu.ConstantGammaClutter('Sensor',ha,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'PRF',prf,...
    'SampleRate',fs,'Gamma',tergamma,'EarthModel','Flat',...
    'TransmitERP',tpower,'PlatformHeight',height,...
    'PlatformSpeed',speed,'PlatformDirection',direction,...
    'BroadsideDepressionAngle',depang,'MaximumRange',Rmax,...
    'AzimuthCoverage',Azcov);
```

Simulate the clutter return for 10 pulses.

```
Nsamp = fs/prf; Npulse = 10;
csig = zeros(Nsamp,Nele,Npulse);
for m = 1:Npulse
    csig(:,:,m) = step(hclutter);
end
```

Plot the angle-Doppler response of the clutter at the 20th range bin.

```
hresp = phased.AngleDopplerResponse('SensorArray',ha,...
    'OperatingFrequency',fc,'PropagationSpeed',c,'PRF',prf);
plotResponse(hresp,shiftdim(csig(20,:,:)),...
    'NormalizeDoppler',true);
```



The results do not exactly match those achieved by using `phased.ConstantGammaClutter` instead of `phased.gpu.ConstantGammaClutter`. This discrepancy occurs because of differences between CPU and GPU computations.

Clutter Simulation Using Known Transmit Signal

Simulate the clutter return from terrain with a gamma value of 0 dB. The `step` syntax includes the transmit signal of the radar system as an input argument. In this case, you do not record the effective transmitted power of the signal in a property.

Set up the characteristics of the radar system. This system has a 4-element uniform linear array (ULA). The sample rate is 1 MHz, and the PRF is 10 kHz. The propagation speed is 300,000 km/s, and the operating frequency is 300 MHz. The radar platform is flying 1 km above the ground with a path parallel to the ground along the array axis. The platform speed is 2000 m/s. The mainlobe has a depression angle of 30 degrees.

```
Nele = 4;
c = 3e8; fc = 3e8; lambda = c/fc;
ha = phased.ULA('NumElements',Nele,'ElementSpacing',lambda/2);

fs = 1e6; prf = 10e3;
height = 1000; direction = [90; 0];
speed = 2000; depang = 30;
```

Create the GPU clutter simulation object and configure it to take a transmit signal as an input argument to `step`. The configuration assumes the earth is flat. The maximum clutter range of interest is 5 km, and the maximum azimuth coverage is +/– 60 degrees.

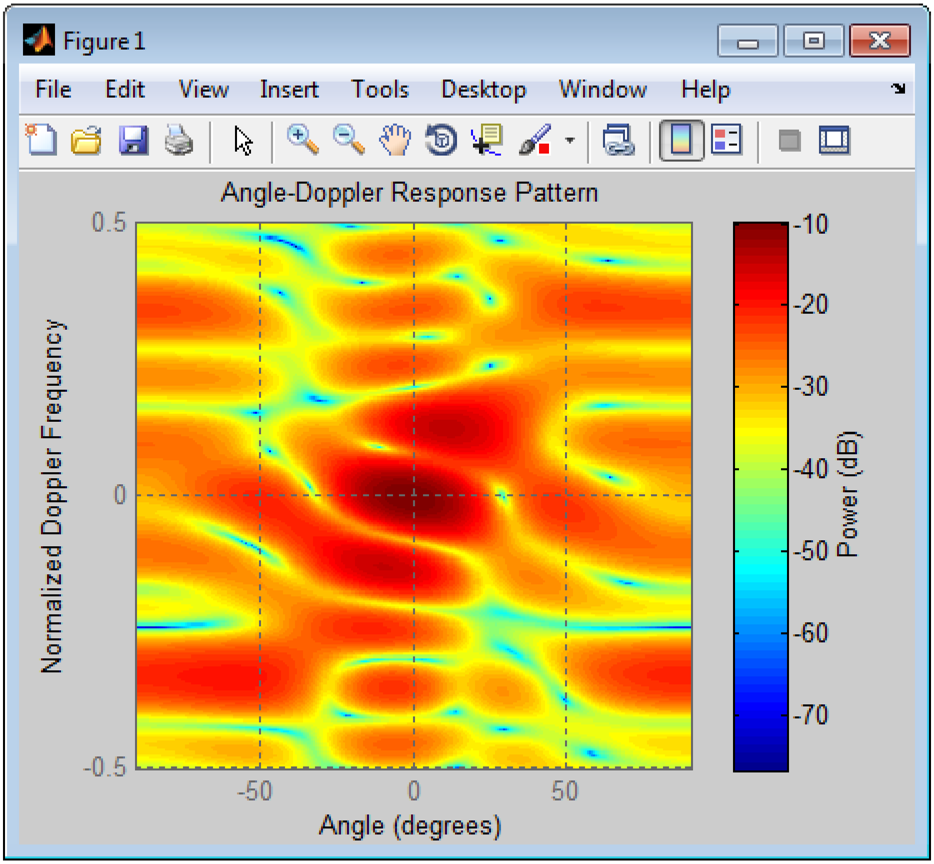
```
Rmax = 5000; Azcov = 120;
tergamma = 0;
hclutter = phased.gpu.ConstantGammaClutter('Sensor',ha,...
    'PropagationSpeed',c,'OperatingFrequency',fc,'PRF',prf,...
    'SampleRate',fs,'Gamma',tergamma,'EarthModel','Flat',...
    'TransmitSignalInputPort',true,'PlatformHeight',height,...
    'PlatformSpeed',speed,'PlatformDirection',direction,...
    'BroadsideDepressionAngle',depang,'MaximumRange',Rmax,...
    'AzimuthCoverage',Azcov);
```

Simulate the clutter return for 10 pulses. At each step, pass the transmit signal as an input argument. The software automatically computes the effective transmitted power of the signal. The transmit signal is a rectangular waveform with a pulse width of 2 μs.

```
tpower = 5000;
pw = 2e-6;
X = tpower*ones(floor(pw*fs),1);
Nsamp = fs/prf; Npulse = 10;
csig = zeros(Nsamp,Nele,Npulse);
for m = 1:Npulse
    csig(:,:,m) = step(hclutter,X);
end
```

Plot the angle-Doppler response of the clutter at the 20th range bin.

```
hresp = phased.AngleDopplerResponse('SensorArray',ha,...
    'OperatingFrequency',fc,'PropagationSpeed',c,'PRF',prf);
plotResponse(hresp,shiftdim(csig(20,:,:)),...
    'NormalizeDoppler',true);
```



The results do not exactly match those achieved by using `phased.ConstantGammaClutter` instead of `phased.gpu.ConstantGammaClutter`. This discrepancy occurs because of differences between CPU and GPU computations.

Related Examples

- GPU Acceleration of Clutter Simulation
- Ground Clutter Mitigation with Moving Target Indication (MTI) Radar

Concepts

- “Clutter Modeling”
- “GPU Computing”

phased.IsotropicAntennaElement

Purpose

Isotropic antenna

Description

The `IsotropicAntennaElement` object creates an antenna element with an isotropic response pattern.

To compute the response of the antenna element for specified directions:

1. Define and set up your isotropic antenna element. See "Construction" on page 3-386.
2. Call `step` to compute the antenna response according to the properties of `phased.IsotropicAntennaElement`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.IsotropicAntennaElement` creates an isotropic antenna system object, `H`. The object models an antenna element whose response is 1 in all directions.

`H = phased.IsotropicAntennaElement(Name,Value)` creates an isotropic antenna 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

FrequencyRange

Operating frequency range

Specify the antenna element operating frequency range (in hertz) as a 1-by-2 row vector in the form of `[LowerBound HigherBound]`. The default value of this property represents the UHF band. The antenna element has 0 response outside the specified frequency range.

Default: `[3e8 1e9]`

BackBaffled

Baffle the back of antenna element

Set this property to `true` to baffle the back of the antenna element. In this case, the antenna responses to all azimuth angles beyond +/– 90 degrees from the broadside (0 degrees azimuth and elevation) are 0.

When the value of this property is `false`, the back of the antenna element is not baffled.

Default: `false`

Methods

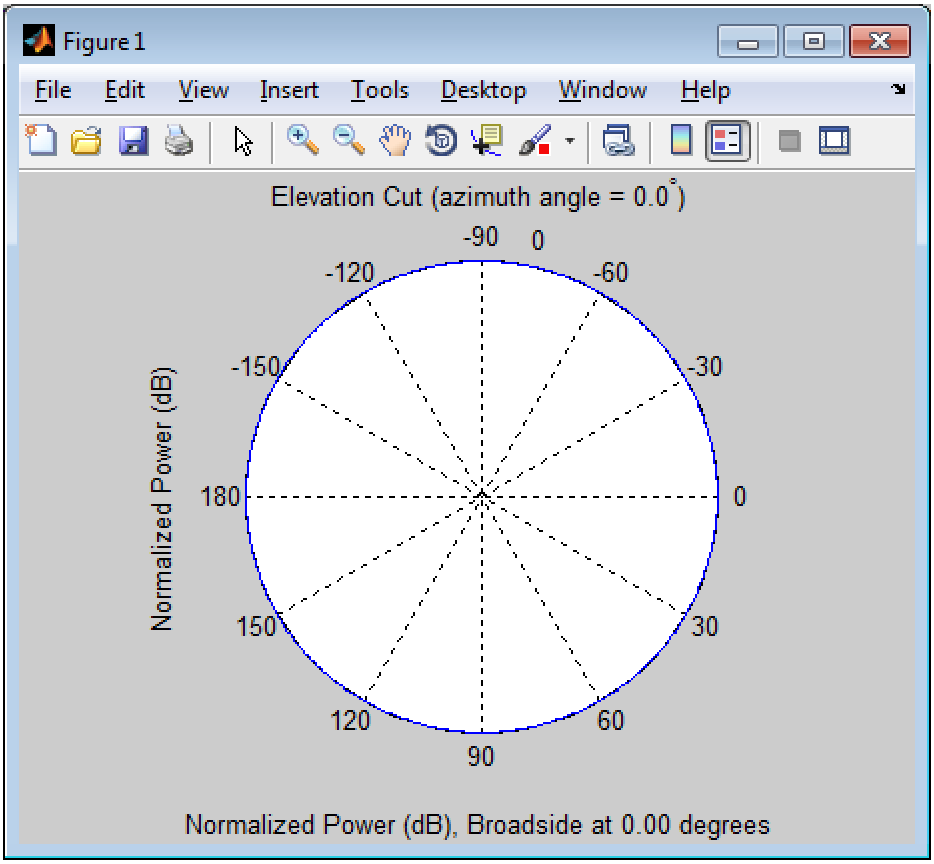
clone	Create isotropic antenna object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of antenna
release	Allow property value and input characteristics changes
step	Output response of antenna element

Examples

Construct 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 the boresight. Then, plot the polar-pattern elevation response of the antenna.

```
ha = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
resp = step(ha,fc,[0; 0]);
```

```
plotResponse(ha,fc,'RespCut','El','Format','Polar');
```



See Also

`phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.ULA` | `phased.URA` |

Purpose Create isotropic antenna object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.IsotropicAntennaElement.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.IsotropicAntennaElement.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.IsotropicAntennaElement.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `IsotropicAntennaElement` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.IsotropicAntennaElement.plotResponse

Purpose

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 object.

FREQ

Operating frequency in hertz. If `FREQ` is a nonscalar row vector, the plot shows multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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

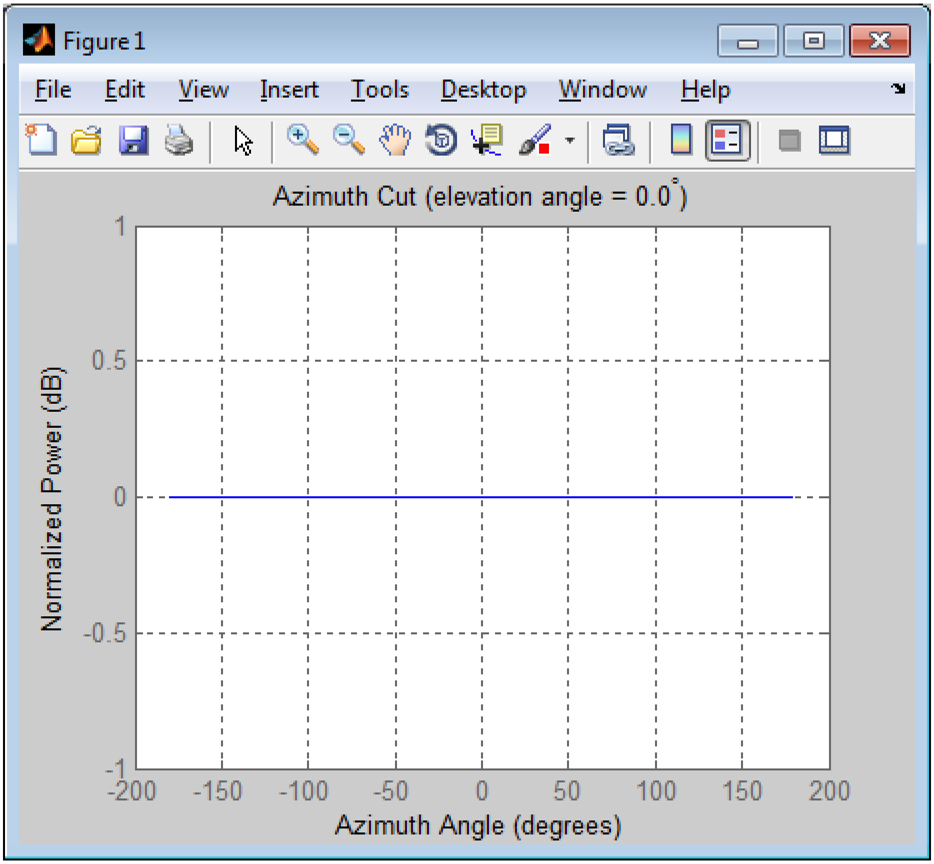
The unit of the plot. Valid values are `'db'`, `'mag'`, and `'pow'`.

Default: `'db'`

Examples

Plot the azimuth cut response of an isotropic antenna along 0 elevation using a line plot. Assume the operating frequency is 1 GHz.

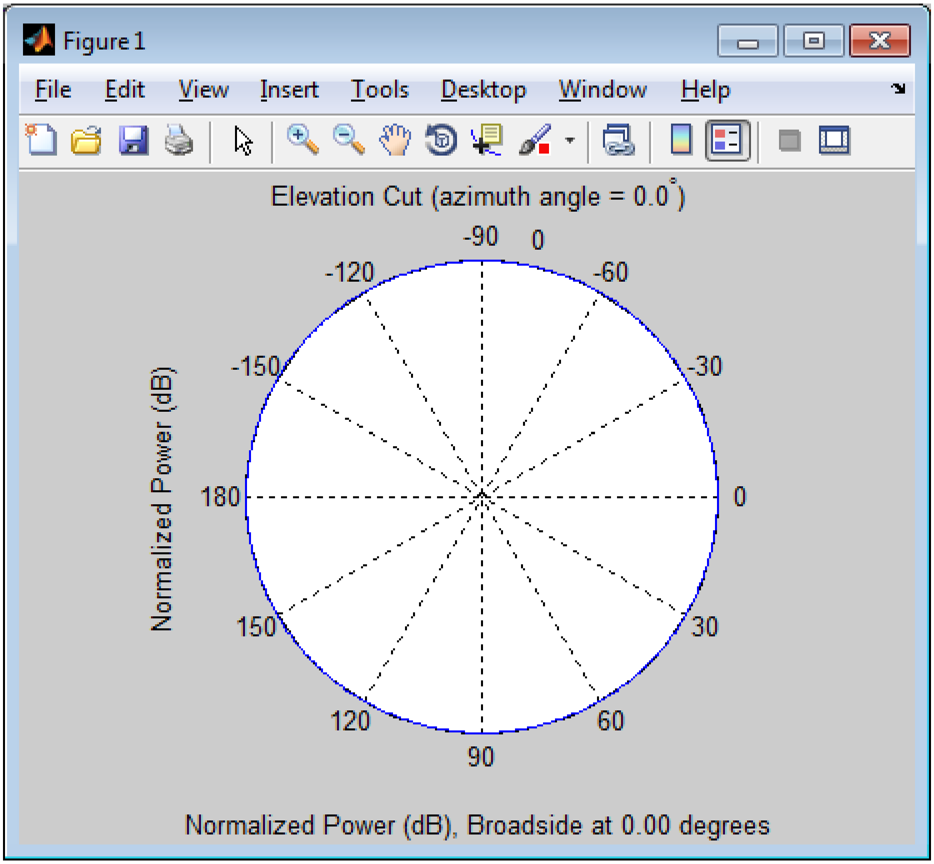
```
ha = phased.IsotropicAntennaElement;
plotResponse(ha,1e9)
```



Construct 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 the boresight. Then, plot the polar-pattern elevation response of the antenna.

```
ha = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
resp = step(ha,fc,[0; 0]);
```

```
plotResponse(ha,fc,'RespCut','El','Format','Polar');
```



See Also

`uv2azel` | `azel2uv`

phased.IsotropicAntennaElement.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Output response of antenna element

Syntax

`RESP = step(H,FREQ,ANG)`

Description

`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 `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

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

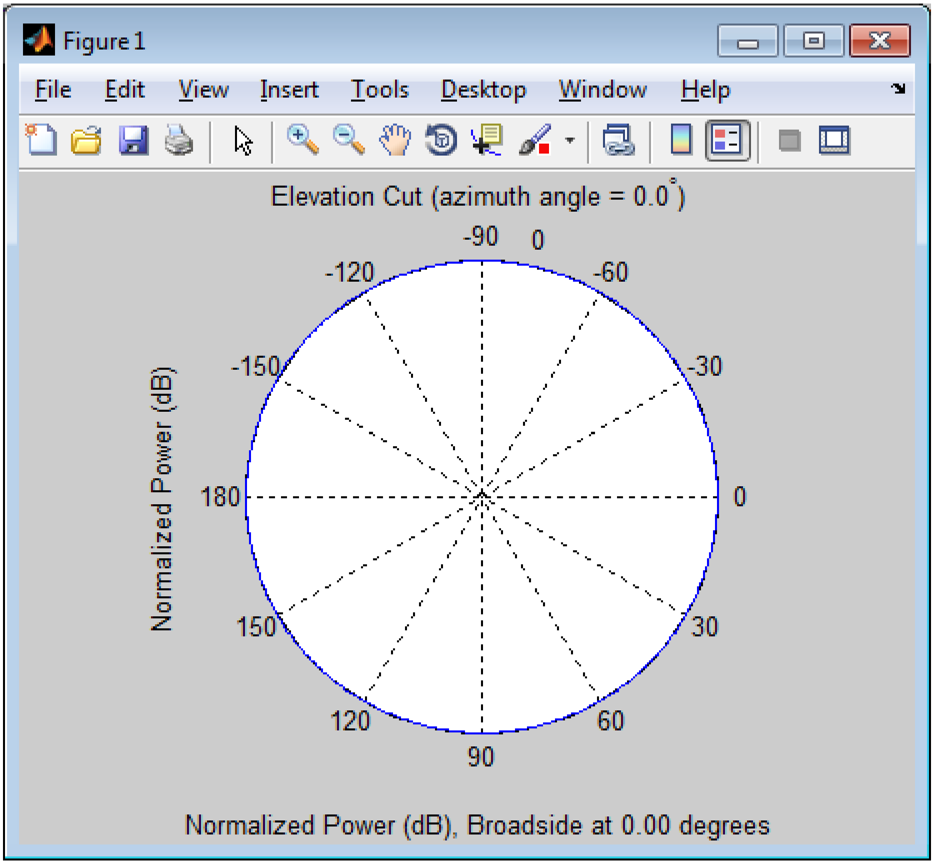
RESP

Voltage response of antenna element. `RESP` is an M-by-L matrix. `RESP` contains the responses at the M angles specified in `ANG` and the L frequencies specified in `FREQ`.

Examples

Construct 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 the boresight. Then, plot the polar-pattern elevation response of the antenna.

```
ha = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
resp = step(ha,fc,[0; 0]);
plotResponse(ha,fc,'RespCut','El','Format','Polar');
```



See Also `uv2azel` | `phitheta2azel`

Purpose

Narrowband LCMV beamformer

Description

The `LCMVBeamformer` object implements a linear constraint minimum variance beamformer.

To compute the beamformed signal:

1. Define and set up your LCMV beamformer. See "Construction" on page 3-402.
2. Call `step` to perform the beamforming operation according to the properties of `phased.LCMVBeamformer`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.LCMVBeamformer` creates a linear constraint minimum variance (LCMV) beamformer System object, `H`. The object performs narrowband LCMV beamforming on the received signal.

`H = phased.LCMVBeamformer(Name,Value)` creates an 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

Constraint

Constraint matrix

Specify the constraint matrix used for LCMV beamforming as an N-by-K matrix. Each column of the matrix is a constraint and N is the number of elements in the sensor array.

Default: `[1; 1]`

DesiredResponse

Desired response vector

Specify the desired response used for LCMV beamforming 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.

Default: `1`, which corresponds 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.

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`

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

clone	Create LCMV beamformer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform LCMV beamforming

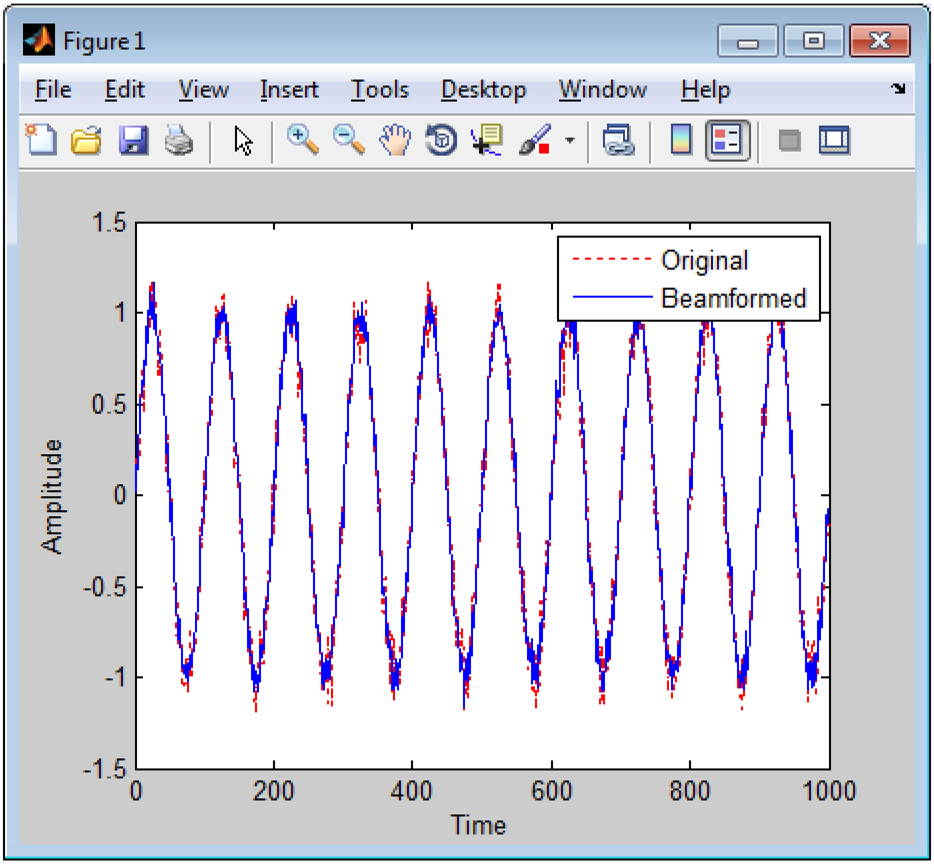
Examples

Apply an LCMV beamformer to a 5-element ULA, preserving the signal from the desired direction.

```
% Simulate signal
t = (0:1000)';
x = sin(2*pi*0.01*t);
c = 3e8; Fc = 3e8;
incidentAngle = [45; 0];
ha = phased.ULA('NumElements',5);
x = collectPlaneWave(ha,x,incidentAngle,Fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;

% Beamforming
hstv = phased.SteeringVector('SensorArray',ha,...
    'PropagationSpeed',c);
hbf = phased.LCMVBeamformer;
hbf.Constraint = step(hstv,Fc,incidentAngle);
hbf.DesiredResponse = 1;
y = step(hbf, rx);
```

```
% Plot
plot(t,real(rx(:,3)),'r:',t,real(y));
xlabel('Time'); ylabel('Amplitude');
legend('Original','Beamformed');
```



References

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

See Also

`phased.MVDRBeamformer` | `phased.PhaseShiftBeamformer` |
`phased.TimeDelayLCMVBeamformer` |

Concepts

- "Adaptive Beamforming"

Purpose

Create LCMV beamformer object with same property values

Syntax

`C = clone(H)`

Description

`C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.LCMVBeamformer.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.LCMVBeamformer.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `LCMVBeamformer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.LCMVBeamformer.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.LCMVBeamformer.step

Purpose

Perform LCMV beamforming

Syntax

```
Y = step(H,X)
Y = step(H,X,XT)
[Y,W] = step(___)
```

Description

`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.

`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.

`[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 `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.

Examples

Apply an LCMV beamformer to a 5-element ULA, preserving the signal from the desired direction.

```
% Simulate signal
t = (0:1000)';
x = sin(2*pi*0.01*t);
c = 3e8; Fc = 3e8;
```

```
incidentAngle = [45; 0];
ha = phased.ULA('NumElements',5);
x = collectPlaneWave(ha,x,incidentAngle,Fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;

% Beamforming
hstv = phased.SteeringVector('SensorArray',ha,...
    'PropagationSpeed',c);
hbf = phased.LCMVBeamformer;
hbf.Constraint = step(hstv,Fc,incidentAngle);
hbf.DesiredResponse = 1;
y = step(hbf, rx);
```

Purpose

Linear FM pulse waveform

Description

The `LinearFMWaveform` object creates a linear FM pulse waveform.

To obtain waveform samples:

1. Define and set up your linear FM waveform. See "Construction" on page 3-414.
2. Call `step` to generate the linear FM waveform samples according to the properties of `phased.LinearFMWaveform`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.LinearFMWaveform` creates a linear FM pulse waveform System object, `H`. The object generates samples of a linear FM pulse waveform.

`H = phased.LinearFMWaveform(Name,Value)` creates a linear FM 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

Specify the sample rate, in hertz, as a positive scalar. The quantity `(SampleRate ./ PRF)` is a scalar or vector that must contain only integers. The default value of this property corresponds to 1 MHz.

Default: `1e6`

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`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency (in hertz) as a scalar or a row vector. The default value of this property corresponds to 10 kHz.

To implement a constant PRF, specify `PRF` as a positive scalar. To implement a staggered PRF, specify `PRF` as a row vector with positive elements. When `PRF` is a vector, the output pulses use successive elements of the vector as the PRF. If the last element of the vector is reached, the process continues cyclically with the first element of the vector.

The value of this property must satisfy these constraints:

- `PRF` is less than or equal to `(1/PulseWidth)`.
- `(SampleRate ./ PRF)` is a scalar or vector that contains only integers.

Default: `1e4`

SweepBandwidth

FM sweep bandwidth

Specify the bandwidth of the linear FM sweeping (in hertz) as a positive scalar. The default value corresponds to 100 kHz.

Default: `1e5`

SweepDirection

FM sweep direction

Specify the direction of the linear FM sweep as one of `'Up'` or `'Down'`.

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'`

Envelope

Envelope function

Specify the envelope function as one of `'Rectangular'` or `'Gaussian`'.

Default: `'Rectangular'`

OutputFormat

Output signal format

Specify the format of the output signal as one of `'Pulses'` or `'Samples'`. When you set the `OutputFormat` property to `'Pulses'`, the output of the `step` method is in the form of multiple pulses. In this case, the number of pulses is the value of the `NumPulses` property.

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: `'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`

Methods

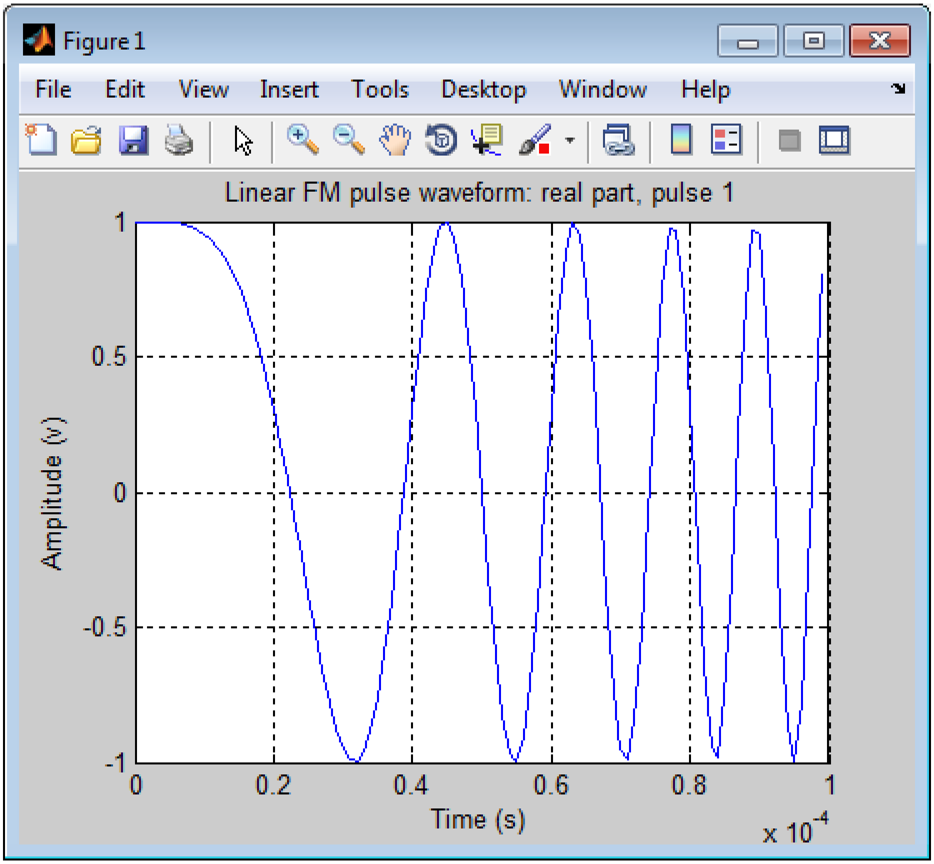
bandwidth	Bandwidth of linear FM waveform
clone	Create linear FM waveform object with same property values
getMatchedFilter	Matched filter coefficients for waveform
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
getStretchProcessor	Create stretch processor for waveform
isLocked	Locked status for input attributes and nontunable properties
plot	Plot linear FM pulse waveform
release	Allow property value and input characteristics changes

reset	Reset states of the linear FM waveform object
step	Samples of linear FM pulse waveform

Examples

Create and plot an upsweep linear FM pulse waveform.

```
hw = phased.LinearFMWaveform('SweepBandwidth',1e5,...
    'PulseWidth',1e-4);
plot(hw);
```



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.

See Also

`phased.RectangularWaveform` | `phased.SteppedFMWaveform` | `phased.PhaseCodedWaveform` |

Related Examples

- Waveform Analysis Using the Ambiguity Function

Purpose

Bandwidth of linear FM waveform

Syntax

```
BW = bandwidth(H)
```

Description

`BW = bandwidth(H)` returns the bandwidth (in hertz) of the pulses for the linear FM pulse waveform `H`. The bandwidth equals the value of the `SweepBandwidth` property.

Input Arguments

H

Linear FM pulse waveform object.

Output Arguments

BW

Bandwidth of the pulses, in hertz.

Examples

Determine the bandwidth of a linear FM pulse waveform.

```
H = phased.LinearFMWaveform;
bw = bandwidth(H)
```

phased.LinearFMWaveform.clone

Purpose Create linear FM waveform object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.LinearFMWaveform.getMatchedFilter

Purpose Matched filter coefficients for waveform

Syntax

```
Coeff = getMatchedFilter(H)
```

Description `Coeff = getMatchedFilter(H)` returns the matched filter coefficients for the linear FM waveform object `H`. `Coeff` is a column vector.

Examples Get the matched filter coefficients for a linear FM pulse.

```
hwav = phased.LinearFMWaveform('PulseWidth',5e-05,...
    'SweepBandwidth',1e5,'OutputFormat','Pulses');
coeff = getMatchedFilter(hwav);
stem(real(coeff));
title('Matched filter coefficients, real part');
```

phased.LinearFMWaveform.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.LinearFMWaveform.getStretchProcessor

Purpose

Create stretch processor for waveform

Syntax

```
HS = getStretchProcessor(H)
HS = getStretchProcessor(H,refrng)
HS = getStretchProcessor(H,refrng,rngspan)
HS = getStretchProcessor(H,refrng,rngspan,v)
```

Description

`HS = getStretchProcessor(H)` returns the stretch processor for the waveform, `H`. `HS` 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.

`HS = getStretchProcessor(H,refrng)` specifies the reference range.

`HS = getStretchProcessor(H,refrng,rngspan)` specifies the range span. The reference interval is centered at `refrng`.

`HS = getStretchProcessor(H,refrng,rngspan,v)` specifies the propagation speed.

Input Arguments

H

Linear FM pulse waveform object.

refrng

Reference range, in meters, as a positive scalar.

Default: 1/4 of the maximum unambiguous range of a pulse

rngspan

Length of the interval of ranges of interest, in meters, as a positive scalar. The center of the interval is the range value specified in the `refrng` argument.

Default: 1/10 of the distance traveled by the wave within the pulse width

v

Propagation speed, in meters per second, as a positive scalar.

Default: Speed of light

Output Arguments

HS

Stretch processor as a `phased.StretchProcessor` System object.

Examples

Detection of Target Using Stretch Processing

Use stretch processing to locate a target at a range of 4950 m.

Simulate the signal.

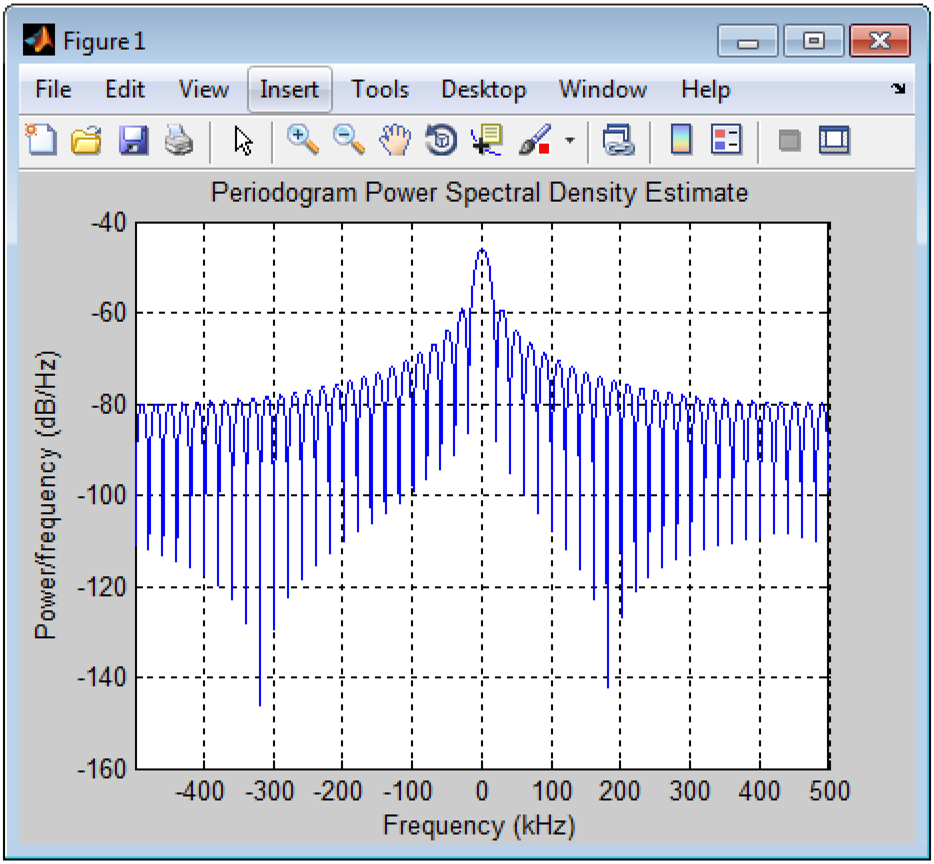
```
hwav = phased.LinearFMWaveform;
x = step(hwav);
c = 3e8; r = 4950;
num_sample = r/(c/(2*hwav.SampleRate));
x = circshift(x,num_sample);
```

Perform stretch processing.

```
hs = getStretchProcessor(hwav,5000,200,c);
y = step(hs,x);
```

Plot the spectrum of the resulting signal.

```
hp = spectrum.periodogram;
hpsd = psd(hp,y,'Fs',hs.SampleRate,'NFFT',2048,...
   'CenterDC',true);
plot(hpsd);
```



Detect the range.

```
[~,rngidx] = findpeaks(pow2db(hpsd.Data/max(hpsd.Data)),...
   'MinPeakHeight',-5);
rngfreq = hpsd.Frequencies(rngidx);
re = stretchfreq2rng(rngfreq,hs.SweepSlope,...
   hs.ReferenceRange,c);
```

See Also

`phased.StretchProcessor` **|** `stretchfreq2rng`

Related Examples

- Range Estimation Using Stretch Processing

Concepts

- “Stretch Processing”

phased.LinearFMWaveform.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `LinearFMWaveform` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.LinearFMWaveform.plot

Purpose

Plot linear 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

String that specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `Type` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Default: `'b'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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`

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

Create and plot an upsweep linear FM pulse waveform.

```
hw = phased.LinearFMWaveform('SweepBandwidth',1e5,...
    'PulseWidth',1e-4);
plot(hw);
```

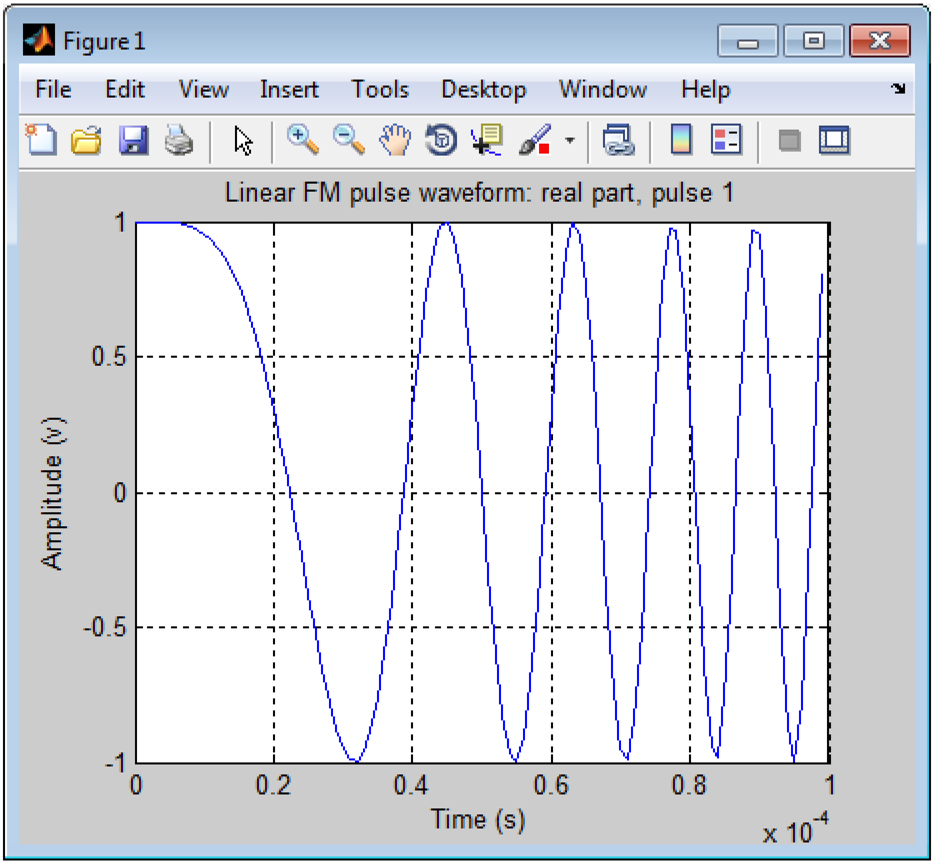


Figure 1
File Edit View Insert Tools Desktop Window Help
Linear FM pulse waveform: real part, pulse 1
Amplitude (v)
Time (s)
x 10^-4
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1

phased.LinearFMWaveform.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose 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.

phased.LinearFMWaveform.step

Purpose

Samples of linear FM pulse waveform

Syntax

```
Y = step(H)
```

Description

`Y = step(H)` returns samples of the linear FM pulse in a column vector `Y`.

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.

Examples

Construct a linear FM waveform with 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.

```
hfmwav = phased.LinearFMWaveform('SweepBandwidth',3e5,...
    'OutputFormat','Pulses','SampleRate',1e6,...
    'PulseWidth',50e-6,'PRF',1e4);
% use step method to obtain the linear FM waveform
wav = step(hfmwav);
```

Purpose

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 3-437.
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.

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

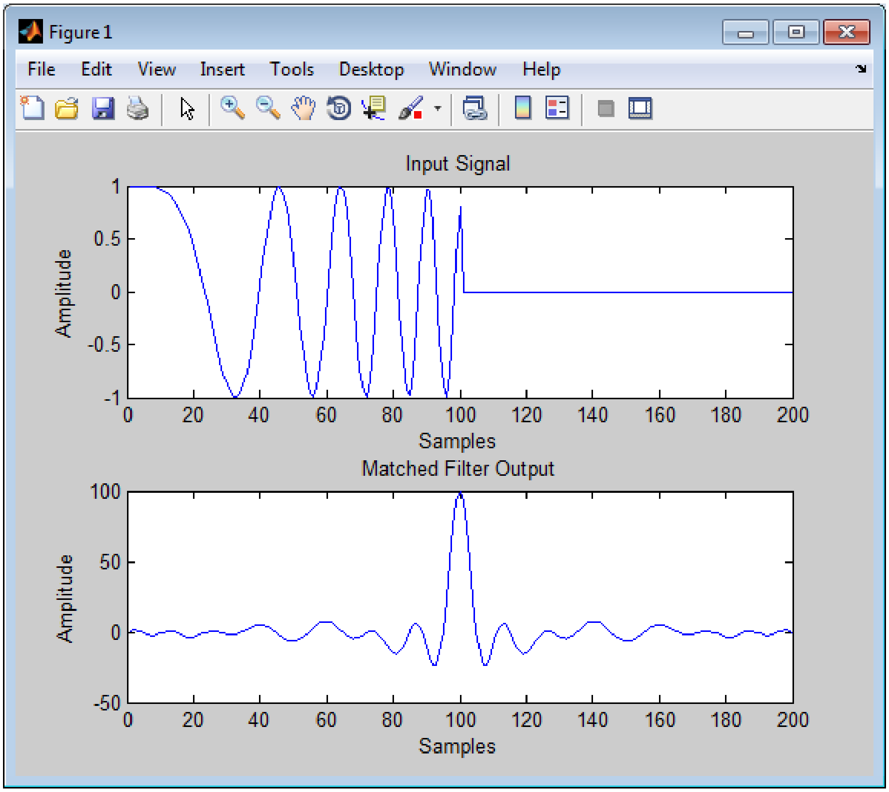
clone	Create matched filter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform matched filtering

Examples

Construct a matched filter for a linear FM waveform.

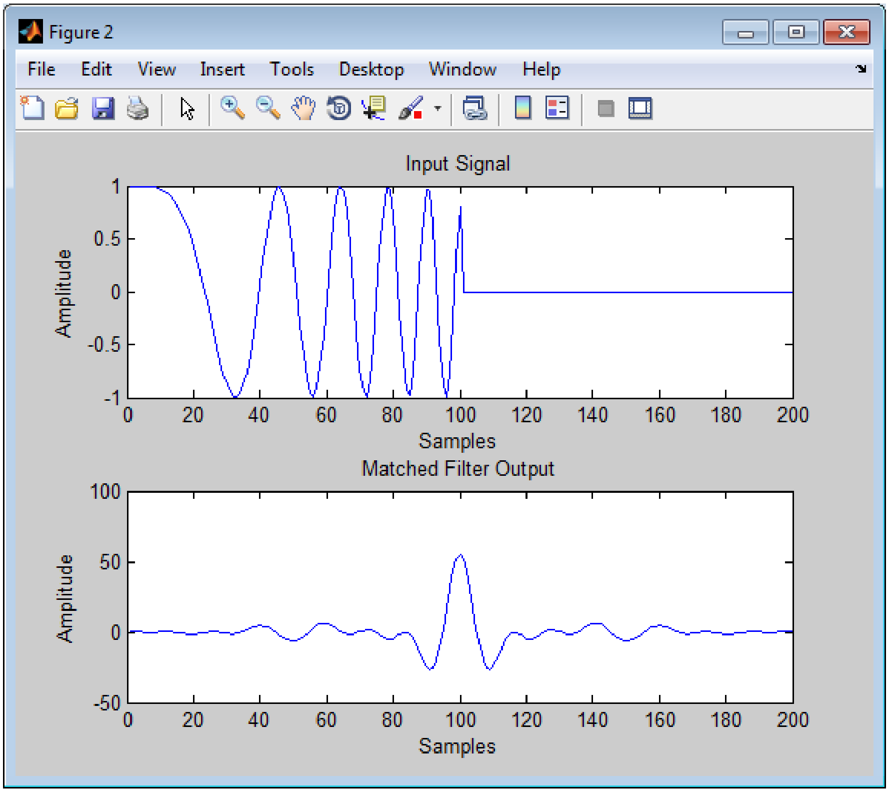
```
hw = phased.LinearFMWaveform('PulseWidth',1e-4,'PRF',5e3);
x = step(hw);
hmf = phased.MatchedFilter(...
    'Coefficients',getMatchedFilter(hw));
y = step(hmf,x);
subplot(211),plot(real(x));
xlabel('Samples'); ylabel('Amplitude');
title('Input Signal');
subplot(212),plot(real(y));
xlabel('Samples'); ylabel('Amplitude');
title('Matched Filter Output');
```



Apply the matched filter, using a Hamming window to do spectrum weighting.

```
hw = phased.LinearFMWaveform('PulseWidth',1e-4,'PRF',5e3);
x = step(hw);
hmf = phased.MatchedFilter(...
    'Coefficients',getMatchedFilter(hw),...
 'SpectrumWindow','Hamming');
y = step(hmf,x);
subplot(211),plot(real(x));
xlabel('Samples'); ylabel('Amplitude');
```

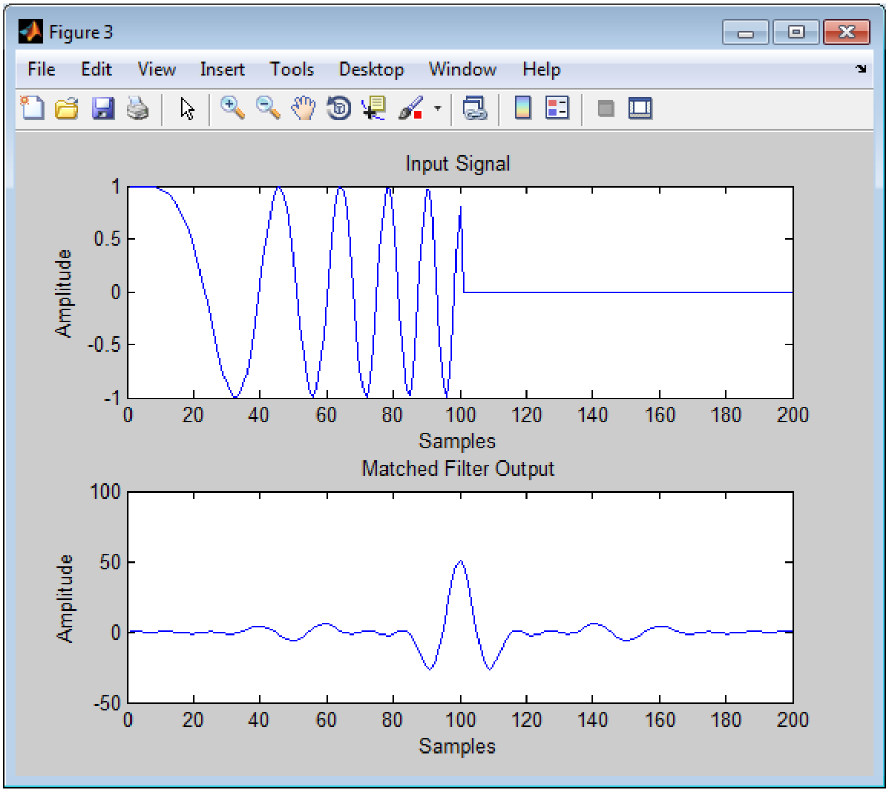
```
title('Input Signal');
subplot(212),plot(real(y));
xlabel('Samples'); ylabel('Amplitude');
title('Matched Filter Output');
```



Apply the matched filter, using a custom Gaussian window for spectrum weighting.

```
hw = phased.LinearFMWaveform('PulseWidth',1e-4,'PRF',5e3);
x = step(hw);
hmf = phased.MatchedFilter(...
```

```
    'Coefficients',getMatchedFilter(hw),...
 'SpectrumWindow','Custom',...
 'CustomSpectrumWindow',{@gausswin,2.5});
y = step(hmf,x);
subplot(211),plot(real(x));
xlabel('Samples'); ylabel('Amplitude');
title('Input Signal');
subplot(212),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].

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

`phased.CFARDetector` **|** `pulsintphased.StretchProcessor` **|** `phased.TimeVaryingGain` **|** | `taylorwin`

phased.MatchedFilter.clone

Purpose Create matched filter object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose

Number of expected inputs to step method

Syntax

`N = getNumInputs(H)`

Description

`N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.MatchedFilter.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `MatchedFilter` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.MatchedFilter.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform matched filtering

Syntax

```
Y = step(H,X)
Y = step(H,X,COEFF)
[Y,GAIN] = step(___)
```

Description

`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.

`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 `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.

Examples

Construct a linear FM waveform with a sweep bandwidth of 300 kHz and a pulse width of 50 microseconds. Obtain the matched filter coefficients using the `getMatchedFilter` method. Use the `step` method for `phased.MatchedFilter` to obtain the matched filter output.

```
hfmwav = phased.LinearFMWaveform('SweepBandwidth',3e5,...
    'OutputFormat','Pulses','SampleRate',1e6,...
    'PulseWidth',50e-6,'PRF',1e4);
% use step method of phased.LinearFMWaveform
```

```
% to obtain the linear FM waveform
wav = step(hfmwav);
% get matched filter coefficients for linear FM waveform
mfcoeffs = getMatchedFilter(hfmwav);
hmf = phased.MatchedFilter('Coefficients',mfcoeffs);
% use step method of phased.MatchedFilter to obtain matched filter
% output
mfoutput = step(hmf,wav);
```

Purpose

Narrowband MVDR (Capon) beamformer

Description

The `MVDRBeamformer` object implements a minimum variance distortionless response beamformer. This is also referred to as a Capon beamformer.

To compute the beamformed signal:

1. Define and set up your MVDR beamformer. See "Construction" on page 3-453.
2. Call `step` to perform the beamforming operation according to the properties of `phased.MVDRBeamformer`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.MVDRBeamformer` creates a minimum variance distortionless response (MVDR) beamformer System object, `H`. The object performs MVDR beamforming on the received signal.

`H = phased.MVDRBeamformer(Name,Value)` creates an MVDR 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 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

OperatingFrequency

System operating frequency

Specify the operating frequency of the beamformer in hertz as a positive scalar. The default value corresponds to 300 MHz.

Default: `3e8`

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.

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 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'`.

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

clone	Create MVDR beamformer object with same property values
getNumInputs	Number of expected inputs to step method

getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform MVDR beamforming

Examples

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.

```
% Signal simulation
t = (0:1000)';
x = sin(2*pi*0.01*t);
c = 3e8; Fc = 3e8;
incidentAngle = [45; 0];
ha = phased.ULA('NumElements',5);
x = collectPlaneWave(ha,x,incidentAngle,Fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x+noise;

% Beamforming
hbf = phased.MVDRBeamformer('SensorArray',ha,...
    'PropagationSpeed',c,'OperatingFrequency',Fc,...
    'Direction',incidentAngle,'WeightsOutputPort',true);
[y,w] = step(hbf,rx);

% Plot signals
plot(t,real(rx(:,3)),'r:',t,real(y));
xlabel('Time'); ylabel('Amplitude');
legend('Original','Beamformed');

% Plot response pattern
figure;
plotResponse(ha,Fc,c,'Weights',w);
```

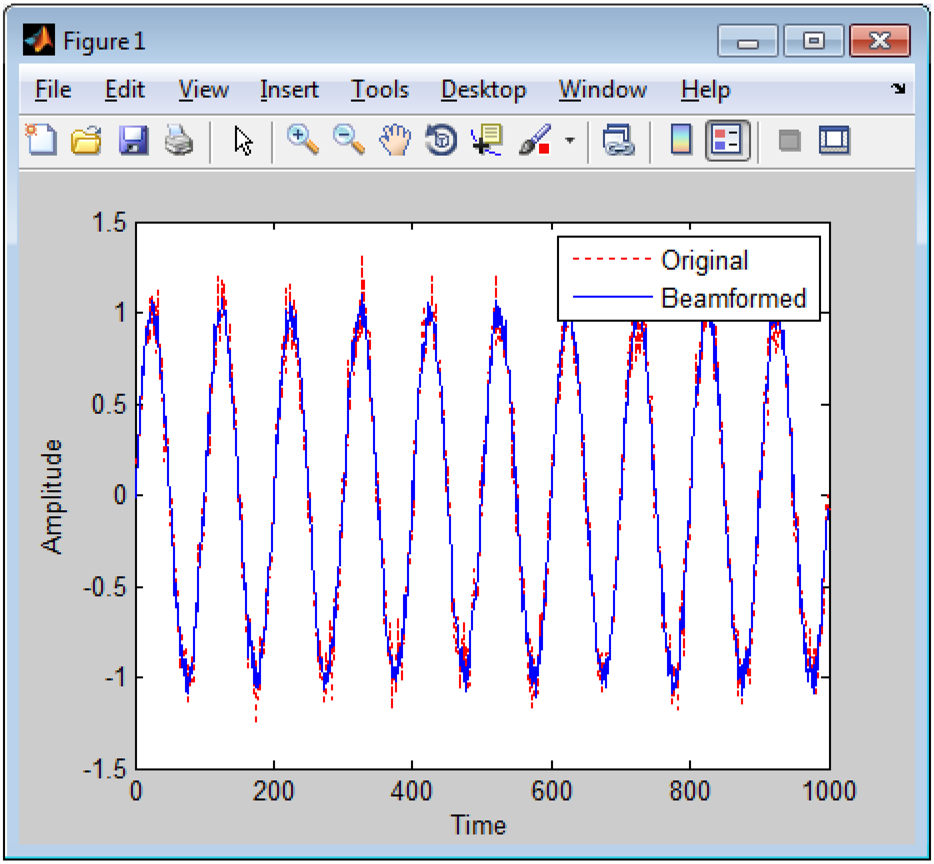
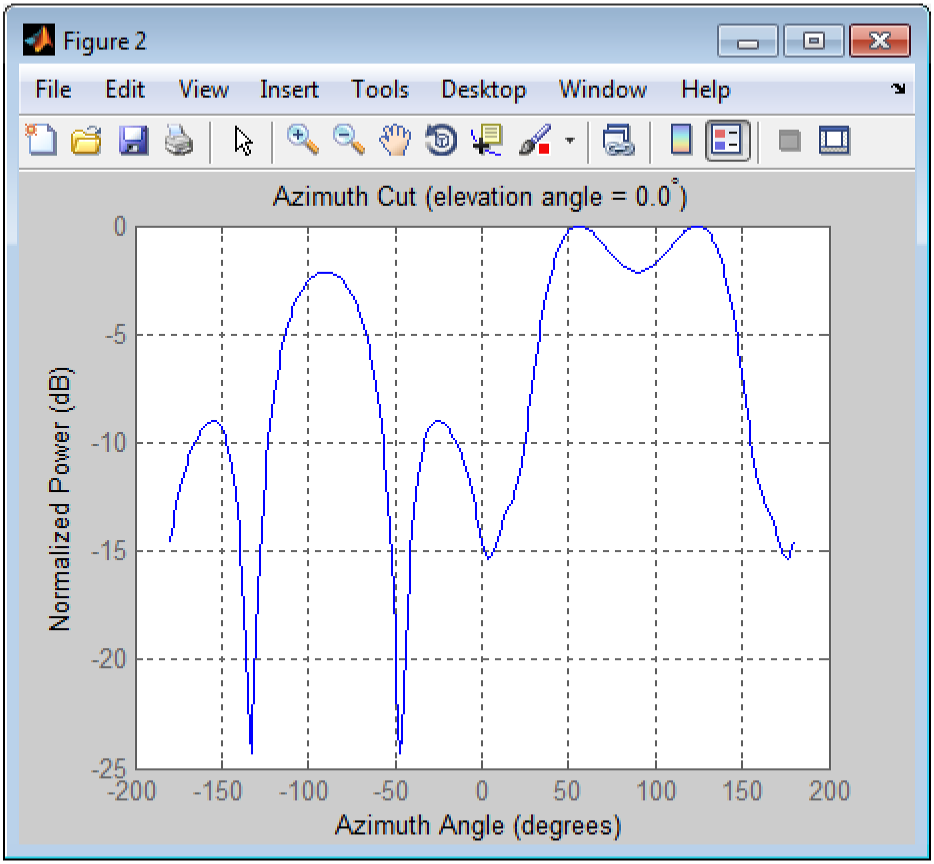


Figure 1
File Edit View Insert Tools Desktop Window Help
Amplitude
Time
1.5
1
0.5
0
-0.5
-1
-1.5
0
200
400
600
800
1000
Original
Beamformed



References

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

See Also

`phased.FrostBeamformer` | `phased.PhaseShiftBeamformer` | `phased.LCMVBeamformer` | `uv2azel` | `phitheta2azel`

Purpose Create MVDR beamformer object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.MVDRBeamformer.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.MVDRBeamformer.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `MVDRBeamformer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.MVDRBeamformer.step

Purpose

Perform MVDR 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

`Y = step(H,X)` performs MVDR beamforming on the input, `X`, and returns the beamformed output in `Y`. This syntax uses `X` as the training samples to calculate the beamforming weights.

`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 `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

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. If you set the `TrainingInputPort` to `false`, M must be larger than N; otherwise, M can be any positive integer.

XT

Training samples, specified as a 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.

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

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.

```
% Signal simulation
```

```
t = (0:1000)';
x = sin(2*pi*0.01*t);
c = 3e8; Fc = 3e8;
incidentAngle = [45; 0];
ha = phased.ULA('NumElements',5);
x = collectPlaneWave(ha,x,incidentAngle,Fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x+noise;

% Beamforming
hbf = phased.MVDRBeamformer('SensorArray',ha,...
    'PropagationSpeed',c,'OperatingFrequency',Fc,...
    'Direction',incidentAngle,'WeightsOutputPort',true);
[y,w] = step(hbf,rx);
```

See Also

`uv2azel` | `phitheta2azel`

Purpose

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 3-467.

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.

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`

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

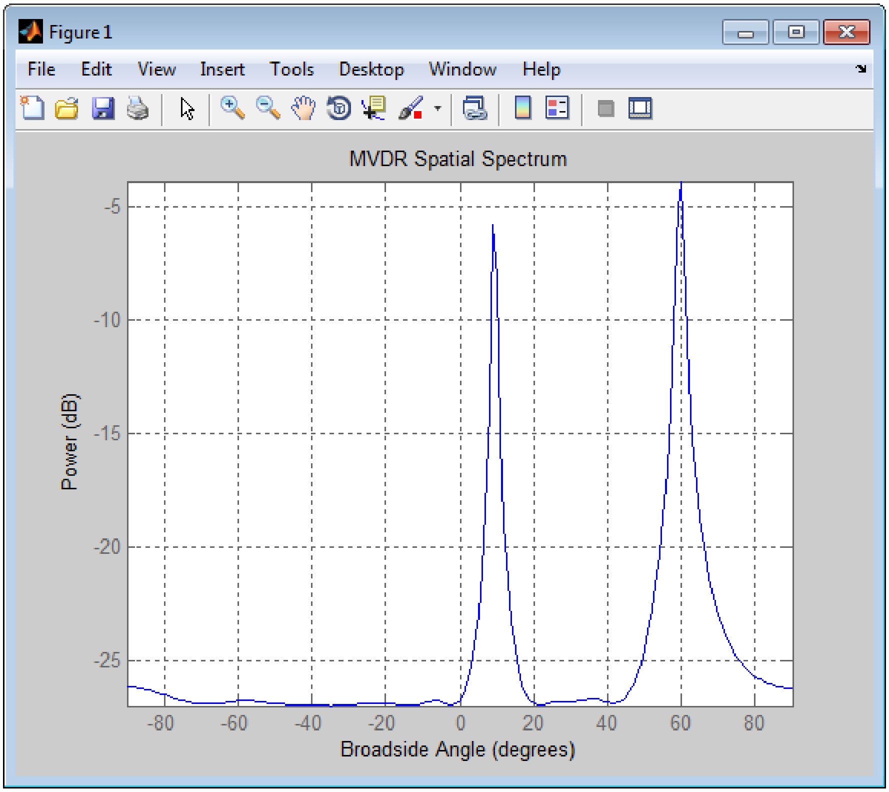
clone	Create MVDR spatial spectrum estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotSpectrum	Plot spatial spectrum
release	Allow property value and input characteristics changes

reset	Reset states of MVDR spatial spectrum estimator object
step	Perform spatial spectrum estimation

Examples

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 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 60 degrees in azimuth and –5 degrees in elevation. This example also plots the spatial spectrum.

```
fs = 8000; t = (0:1/fs:1).';
x1 = cos(2*pi*t*300); x2 = cos(2*pi*t*400);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;60 -5]',fc);
% additive noise
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
% construct MVDR estimator object
hdoa = phased.MVDREstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2);
% use the MVDREstimator step method to obtain the DOA estimates
[y,doas] = step(hdoa,x+noise);
doas = broadside2az(sort(doas),[20 -5]);
plotSpectrum(hdoa);
```



References

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

See Also

`broadside2az` `phased.MVDREstimator2D` |

phased.MVDREstimator.clone

Purpose Create MVDR spatial spectrum estimator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.MVDREstimator.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`N = getNumInputs(H)`

Description

`N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.MVDREstimator.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `MVDREstimator` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.MVDREstimator.plotSpectrum

Purpose

Plot spatial spectrum

Syntax

```
plotSpectrum(H)
plotSpectrum(H,Name,Value)
h = plotSpectrum(___)
```

Description

`plotSpectrum(H)` plots the spatial spectrum resulting from the last call of the `step` method.

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

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

Input Arguments

H

Spatial spectrum estimator object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

NormalizeResponse

Set this value to `true` to plot the normalized spectrum. Set this value to `false` to plot the spectrum without normalizing it.

Default: `false`

Title

String to use as title of figure.

Default: Empty string

Unit

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

Default: `'db'`

Examples

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 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 60 degrees in azimuth and –5 degrees in elevation.

```
fs = 8000; t = (0:1/fs:1).';
x1 = cos(2*pi*t*300); x2 = cos(2*pi*t*400);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;60 -5]',fc);
% additive noise
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
% construct MVDR estimator object
hdoa = phased.MVDREstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2);
% use the MVDREstimator step method to obtain the DOA estimates
[y,doas] = step(hdoa,x+noise);
doas = broadside2az(sort(doas),[20 -5]);
plotSpectrum(hdoa);
```

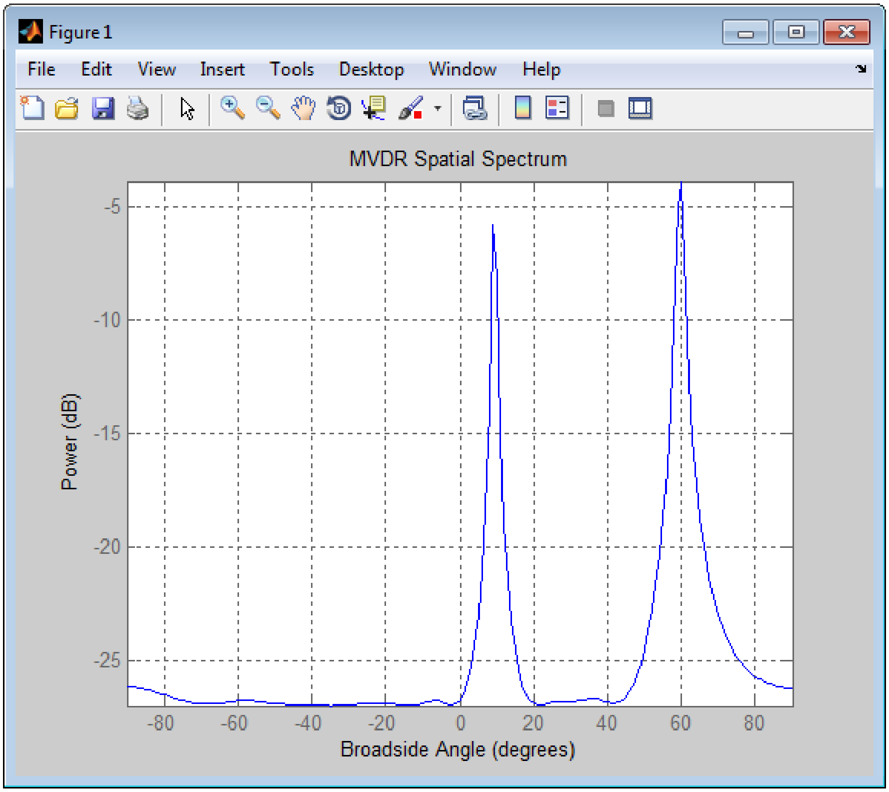


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
MVDR Spatial Spectrum
Power (dB)
-5
-10
-15
-20
-25
-80
-60
-40
-20
0
20
40
60
80
Broadside Angle (degrees)

Purpose

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.MVDREstimator.reset

Purpose Reset states of MVDR spatial spectrum estimator object

Syntax `reset(H)`

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

Purpose

Perform spatial spectrum estimation

Syntax

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

Description

`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.

`[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 `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.

Examples

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 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 60 degrees in azimuth and –5 degrees in elevation.

```
fs = 8000; t = (0:1/fs:1).';
x1 = cos(2*pi*t*300); x2 = cos(2*pi*t*400);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;60 -5]',fc);
% additive noise
```

```
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
% construct MVDR estimator object
hdoa = phased.MVDREstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2);
% use the MVDREstimator step method to obtain the DOA estimates
[y,doas] = step(hdoa,x+noise);
doas = broadside2az(sort(doas),[20 -5]);
```

Purpose

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 3-483.
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.

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.

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`

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.

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.

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`.

Default: `1`

Methods

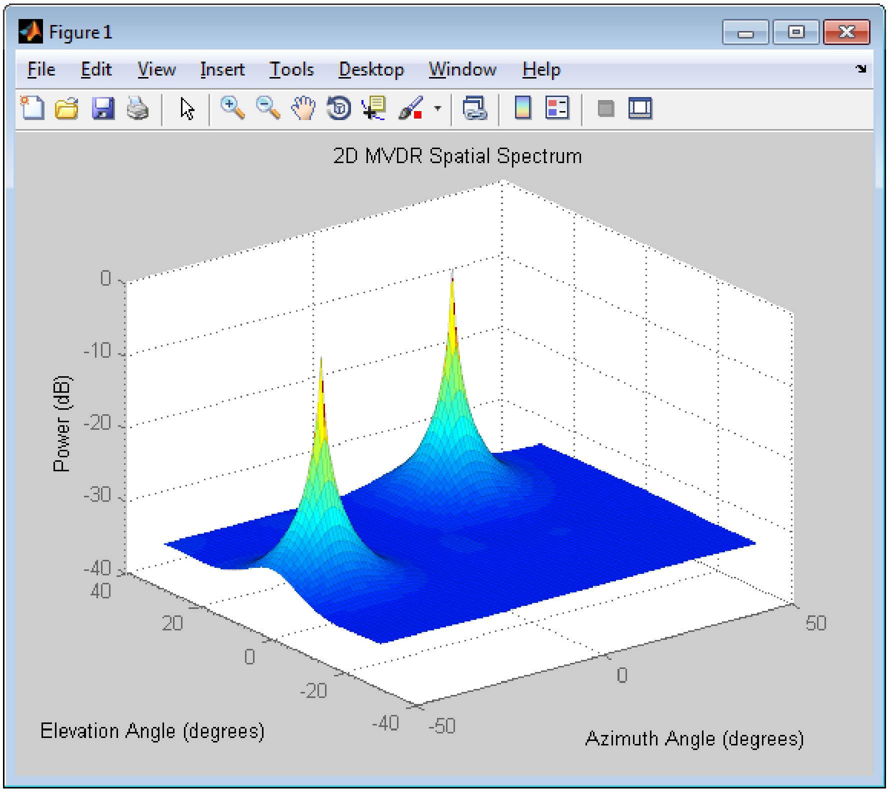
clone	Create 2-D MVDR spatial spectrum estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotSpectrum	Plot spatial spectrum
release	Allow property value and input characteristics changes

reset	Reset states of 2-D MVDR spatial spectrum estimator object
step	Perform spatial spectrum estimation

Examples

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 degrees in azimuth and 0 degrees in elevation. The direction of the second signal is 17 degrees in azimuth and 20 degrees in elevation. This example also plots the spatial spectrum.

```
fs = 8000; t = (0:1/fs:1).';
x1 = cos(2*pi*t*300); x2 = cos(2*pi*t*400);
ha = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[-37 0;17 20]',fc);
% additive noise
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
% construct MVDR DOA estimator for URA
hdoa = phased.MVDREstimator2D('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
% use the step method to obtain the output and DOA estimates
[~,doas] = step(hdoa,x+noise);
plotSpectrum(hdoa);
```



References

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

See Also

`phased.MVDREstimator` | `uv2azel` | `phitheta2azel`

phased.MVDREstimator2D.clone

Purpose Create 2-D MVDR spatial spectrum estimator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.MVDREstimator2D.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `MVDREstimator2D` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.MVDREstimator2D.plotSpectrum

Purpose

Plot spatial spectrum

Syntax

```
plotSpectrum(H)
plotSpectrum(H,Name,Value)
h = plotSpectrum( ___ )
```

Description

`plotSpectrum(H)` plots the spatial spectrum resulting from the last call of the `step` method.

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

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

Input Arguments

H

Spatial spectrum estimator object.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

NormalizeResponse

Set this value to `true` to plot the normalized spectrum. Set this value to `false` to plot the spectrum without normalizing it.

Default: `false`

Title

String to use as title of figure.

Default: Empty string

Unit

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

Default: `'db'`

Examples

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 degrees in azimuth and 0 degrees in elevation. The direction of the second signal is 17 degrees in azimuth and 20 degrees in elevation.

```
fs = 8000; t = (0:1/fs:1).';
x1 = cos(2*pi*t*300); x2 = cos(2*pi*t*400);
ha = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[-37 0;17 20]',fc);
% additive noise
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
% construct MVDR DOA estimator for URA
hdoa = phased.MVDREstimator2D('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
% use the step method to obtain the output and DOA estimates
[~,doas] = step(hdoa,x+noise);
plotSpectrum(hdoa);
```

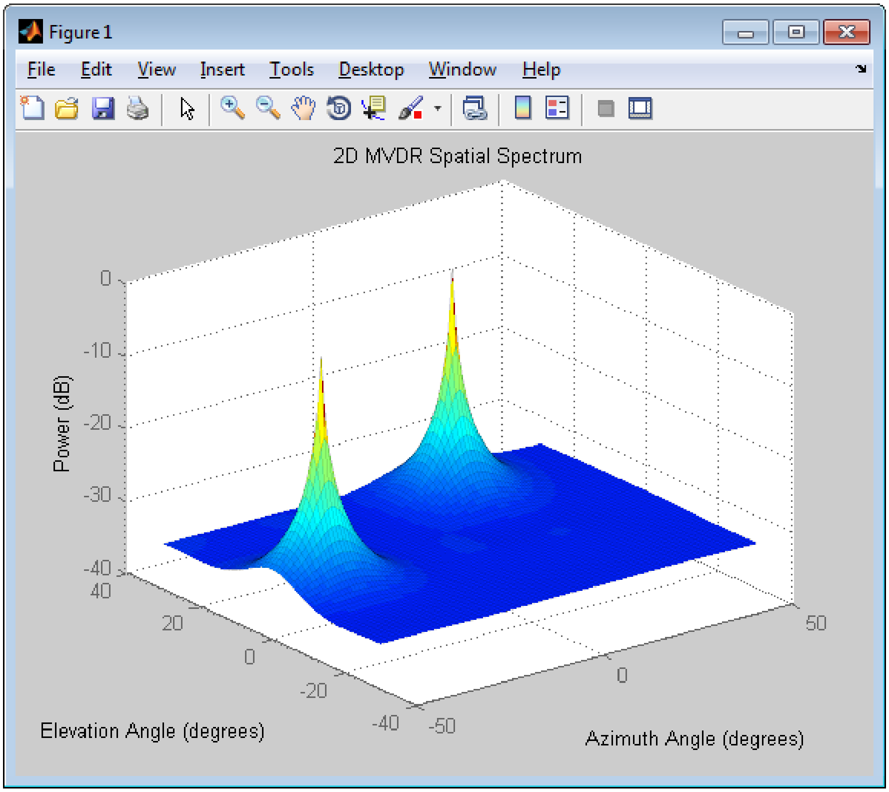


Figure 1
File Edit View Insert Tools Desktop Window Help
2D MVDR Spatial Spectrum
Power (dB)
0
-10
-20
-30
-40
40
20
0
-20
-40
Elevation Angle (degrees)
-50
0
50
Azimuth Angle (degrees)

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.MVDREstimator2D.reset

Purpose Reset states of 2-D MVDR spatial spectrum estimator object

Syntax `reset(H)`

Description `reset(H)` resets the states of the `MVDREstimator2D` object, `H`.

Purpose

Perform spatial spectrum estimation

Syntax

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

Description

`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.

`[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 `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.

Examples

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 degrees in azimuth and 0 degrees in elevation. The direction of the second signal is 17 degrees in azimuth and 20 degrees in elevation.

```
fs = 8000; t = (0:1/fs:1).';
x1 = cos(2*pi*t*300); x2 = cos(2*pi*t*400);
ha = phased.URA('Size',[5 10],'ElementSpacing',[1 0.6]);
ha.Element.FrequencyRange = [100e6 300e6];
```

```
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[-37 0;17 20]',fc);
% additive noise
noise = 0.1*(randn(size(x))+1i*randn(size(x)));
% construct MVDR DOA estimator for URA
hdoa = phased.MVDREstimator2D('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'DOAOutputPort',true,'NumSignals',2,...
    'AzimuthScanAngles',-50:50,...
    'ElevationScanAngles',-30:30);
% use the step method to obtain the output and DOA estimates
[~,doas] = step(hdoa,x+noise);
```

See Also

`azel2uv` | `azel2phitheta`

phased.OmnidirectionalMicrophoneElement

Purpose

Omnidirectional microphone

Description

The `OmnidirectionalMicrophoneElement` object models an omnidirectional microphone with an equal response in all directions.

To compute the response of the microphone element for specified directions:

1. Define and set up your omnidirectional microphone element. See "Construction" on page 3-499.
2. Call `step` to estimate the microphone response according to the properties of `phased.OmnidirectionalMicrophoneElement`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.OmnidirectionalMicrophoneElement` creates an omnidirectional microphone system object, `H`, that models an omnidirectional microphone element whose response is 1 in all directions.

`H = phased.OmnidirectionalMicrophoneElement(Name,Value)` creates an omnidirectional microphone 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

FrequencyRange

Operating frequency range

Specify the operating frequency range (in hertz) of the microphone element as a 1x2 row vector in the form of `[LowerBound HigherBound]`. The default value of this property represents the audible range. The microphone element has no response outside the specified frequency range.

Default: `[20 20e3]`

BackBaffled

Baffle the back of microphone element

Set this property to `true` to baffle the back of the microphone element. In this case, the microphone responses to all azimuth angles beyond +/– 90 degrees from the broadside (0 degree azimuth and elevation) are 0.

When the value of this property is `false`, the back of the microphone element is not baffled.

Default: `false`

Methods

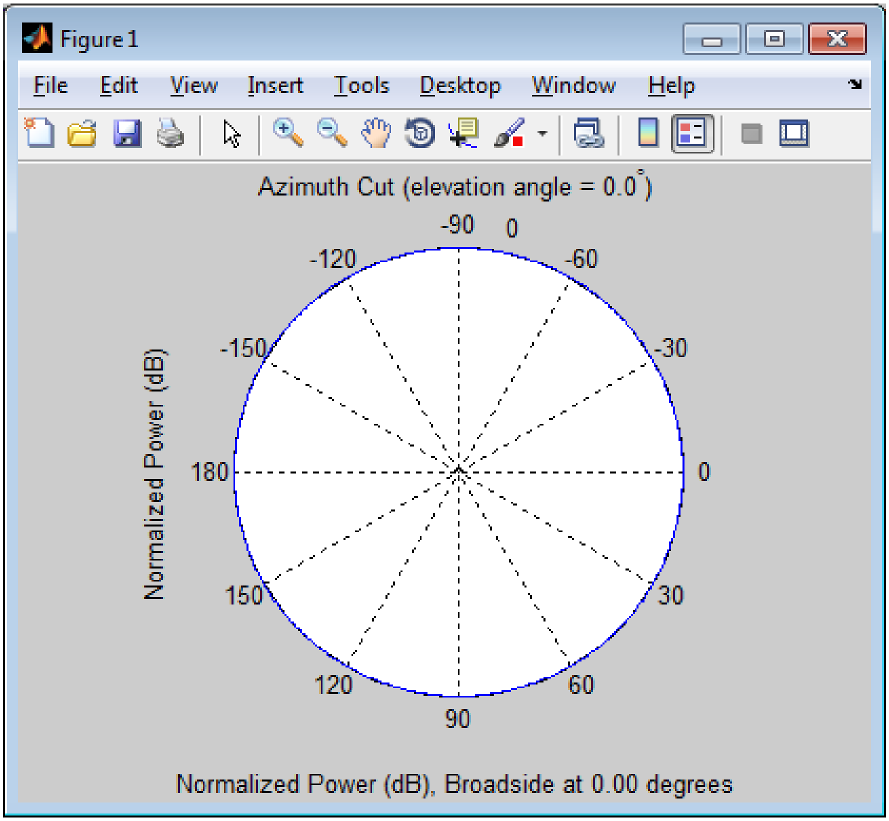
clone	Create omnidirectional microphone object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of microphone
release	Allow property value and input characteristics changes
step	Output response of microphone

Examples

Create an omnidirectional microphone. Find the microphone response at 200, 300, and 400 Hz for the incident angle `[0;0]`. Plot the azimuth response of the microphone.

```
h = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 2e3]);
fc = [200 300 400];
```

```
ang = [0;0];
resp = step(h,fc,ang);
plotResponse(h,200,'RespCut','Az','Format','Polar');
```



See Also

`phased.CustomMicrophoneElement` | `phased.ULA` | `phased.URA` | `phased.ConformalArray` |

phased.OmnidirectionalMicrophoneElement.clone

Purpose Create omnidirectional microphone object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.OmnidirectionalMicrophoneElement.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.OmnidirectionalMicrophoneElement.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.OmnidirectionalMicrophoneElement.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF` of the `OmnidirectionalMicrophoneElement` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.OmnidirectionalMicrophoneElement.plotResponse

Purpose

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 object.

FREQ

Operating frequency in hertz. If `FREQ` is a nonscalar row vector, the plot shows multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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

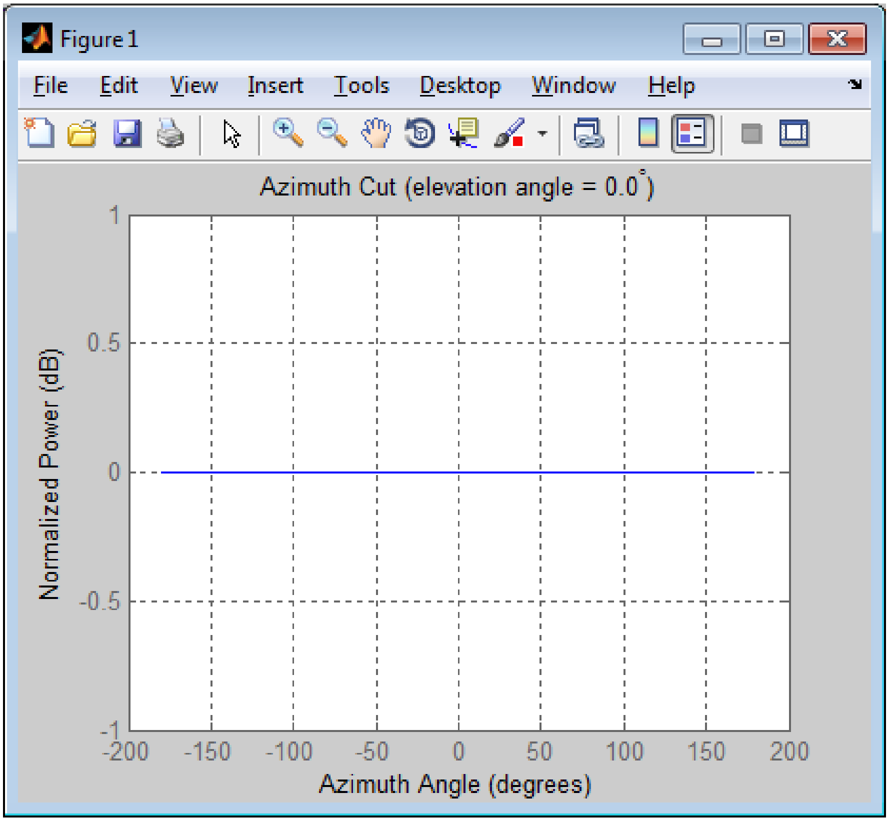
The unit of the plot. Valid values are `'db'`, `'mag'`, and `'pow'`.

Default: `'db'`

Examples

Plot response of omnidirectional microphone.

```
h = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
plotResponse(h,200);
```



See Also `uv2azel` | `azel2uv`

phased.OmnidirectionalMicrophoneElement.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Output response of microphone

Syntax

`RESP = step(H,FREQ,ANG)`

Description

`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 `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

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

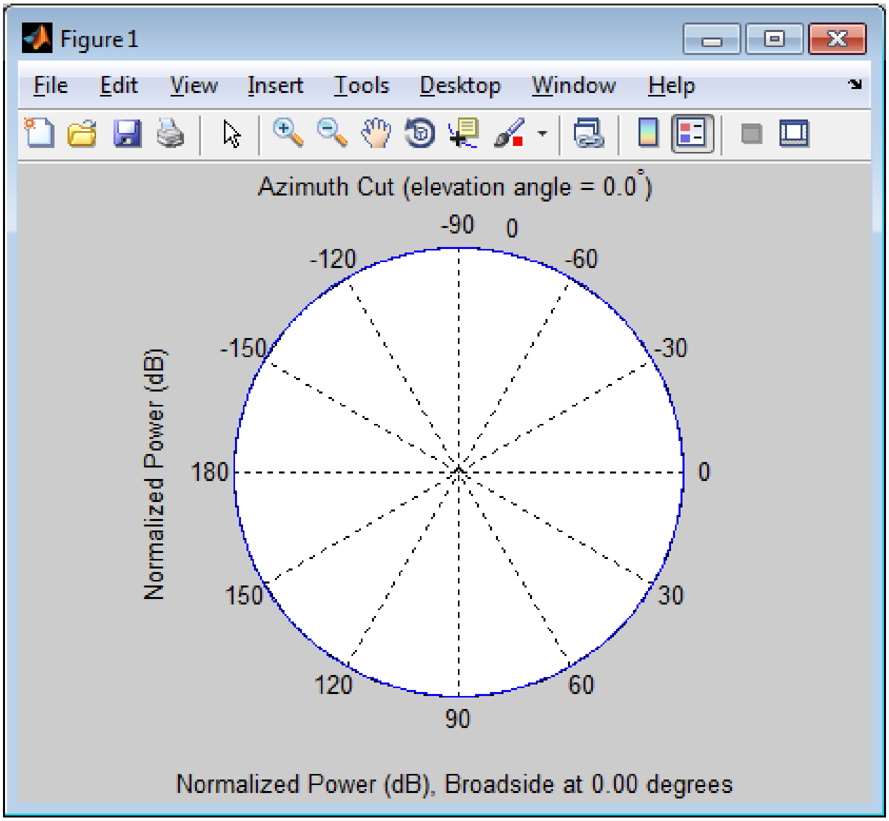
RESP

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

Create an omnidirectional microphone. Find the microphone response at 200, 300, and 400 Hz for the incident angle `[0;0]`. Plot the azimuth response of the microphone.

```
h = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 2e3]);
fc = [200 300 400];
ang = [0;0];
resp = step(h,fc,ang);
plotResponse(h,200,'RespCut','Az','Format','Polar');
```



See Also `uv2azel` | `phitheta2azel`

Purpose

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 3-514.
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.

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

Array aperture

Specify a phased array as a `phased.ULA`, `phased.URA`, or `phased.ConformalArray` object.

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 total number of elements in the array. Each row of the matrix indicates which elements belong to the corresponding subarray. Each entry in the matrix is 1 or 0, where 1 indicates that the element appears in the subarray and 0 indicates the opposite. Each row must contain at least one 1.

The phase center of each subarray is at its 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 steering the subarray as one of `'None' | 'Phase' | 'Time'`.

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`

Methods

clone	Create partitioned array with same property values
collectPlaneWave	Simulate received plane waves
getElementPosition	Positions of array elements

getNumElements	Number of elements in array
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
getNumSubarrays	Number of subarrays in array
getSubarrayPosition	Positions of subarrays in array
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of array
release	Allow property value and input characteristics changes
step	Output responses of subarrays
viewArray	View array geometry

Examples

Azimuth Response of Partitioned ULA

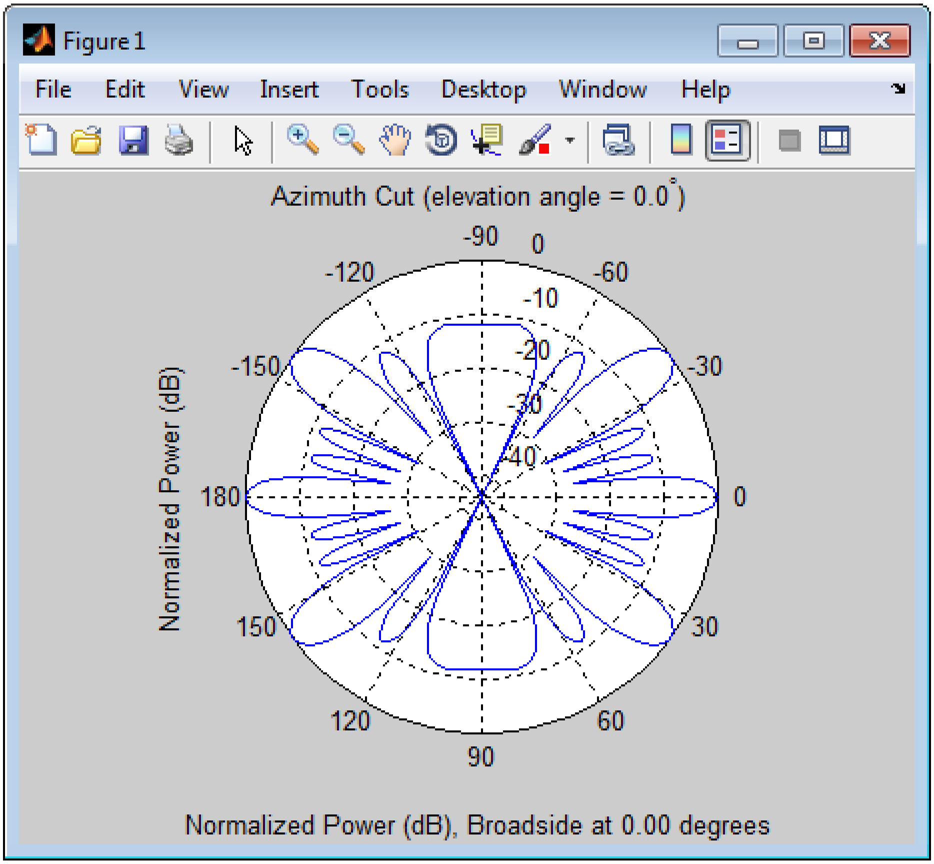
Plot the azimuth response of a 4-element ULA partitioned into two 2-element ULAs.

Create a 4-element ULA, and partition it into 2-element ULAs.

```
h = phased.ULA('NumElements',4,'ElementSpacing',0.5);
ha = phased.PartitionedArray('Array',h,...
   '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 3e8 m/s.

```
plotResponse(ha,1e9,3e8,'RespCut','Az','Format','Polar');
```



Response of Subarrays in Partitioned ULA

Calculate the response at the boresight of a 4-element ULA partitioned into two 2-element ULAs.

Create a 4-element ULA, and partition it into 2-element ULAs.

```
h = phased.ULA('NumElements',4,'ElementSpacing',0.5);
ha = phased.PartitionedArray('Array',h,...
   '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 3e8 m/s.

```
RESP = step(ha,1e9,[0;0],3e8);
```

See Also

`phased.ULA` | `phased.URA` | `phased.ConformalArray` | `phased.ReplicatedSubarray` |

Related Examples

- Subarrays in Phased Array Antennas
- Phased Array Gallery

Concepts

- “Subarrays Within Arrays”

Purpose

Create partitioned array with same property values

Syntax

`C = clone(H)`

Description

`C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.PartitionedArray.collectPlaneWave

Purpose

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)` uses `FREQ` as the incoming signal's carrier frequency.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)` uses `C` as the signal's propagation speed. `C` must be a scalar.

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 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, 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.

```
ha = phased.ULA('NumElements',16);
hpa = phased.PartitionedArray('Array',ha,...
   '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 receiving signals from 10 degrees 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.

```
Y = collectPlaneWave(hpa,randn(4,2),[10 30],...
   1e8,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`

phased.PartitionedArray.getElementPosition

Purpose

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

Positions of Elements in Partitioned Array

Obtain the positions of the six elements in a partitioned array.

```
H = phased.PartitionedArray('Array',phased.URA('Size',[2 3]),...
    'SubarraySelection',[1 0 1 0 1 0; 0 1 0 1 0 1]);
POS = getElementPosition(H);
```

See Also

`getSubarrayPosition` |

phased.PartitionedArray.getNumElements

Purpose

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.

```
H = phased.PartitionedArray('Array',phased.URA('Size',[2 3]),...
   'SubarraySelection',[1 0 1 0 1 0; 0 1 0 1 0 1]);
N = getNumElements(H);
```

See Also

`getNumSubarrays` |

phased.PartitionedArray.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.PartitionedArray.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.PartitionedArray.getNumSubarrays

Purpose 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.

```
H = phased.PartitionedArray('Array',...
   phased.ULA('NumElements',5),...
   'SubarraySelection',[1 1 1 0 0; 0 0 1 1 1]);
N = getNumSubarrays(H);
```

See Also `getNumElements` **|**

phased.PartitionedArray.getSubarrayPosition

Purpose

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

Positions of Subarrays in Partitioned Array

Obtain the positions of the two subarrays in a partitioned array.

```
H = phased.PartitionedArray('Array',phased.URA('Size',[2 3]),...
    'SubarraySelection',[1 0 1 0 1 0; 0 1 0 1 0 1]);
POS = getSubarrayPosition(H);
```

See Also

`getElementPosition` **|**

phased.PartitionedArray.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `PartitionedArray` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.PartitionedArray.plotResponse

Purpose

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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can

specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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'`, and `'pow'`.

Default: `'db'`

Weights

Weights applied to the array, specified as a length-N column vector or N-by-M matrix. N is the number of subarrays in the array. M is the number of frequencies in `FREQ`. If `Weights` is a vector, the function applies the same weights to each frequency. If

`Weights` is a matrix, the function applies each column of weight values to the corresponding frequency in `FREQ`.

Examples

Azimuth Response of Partitioned ULA

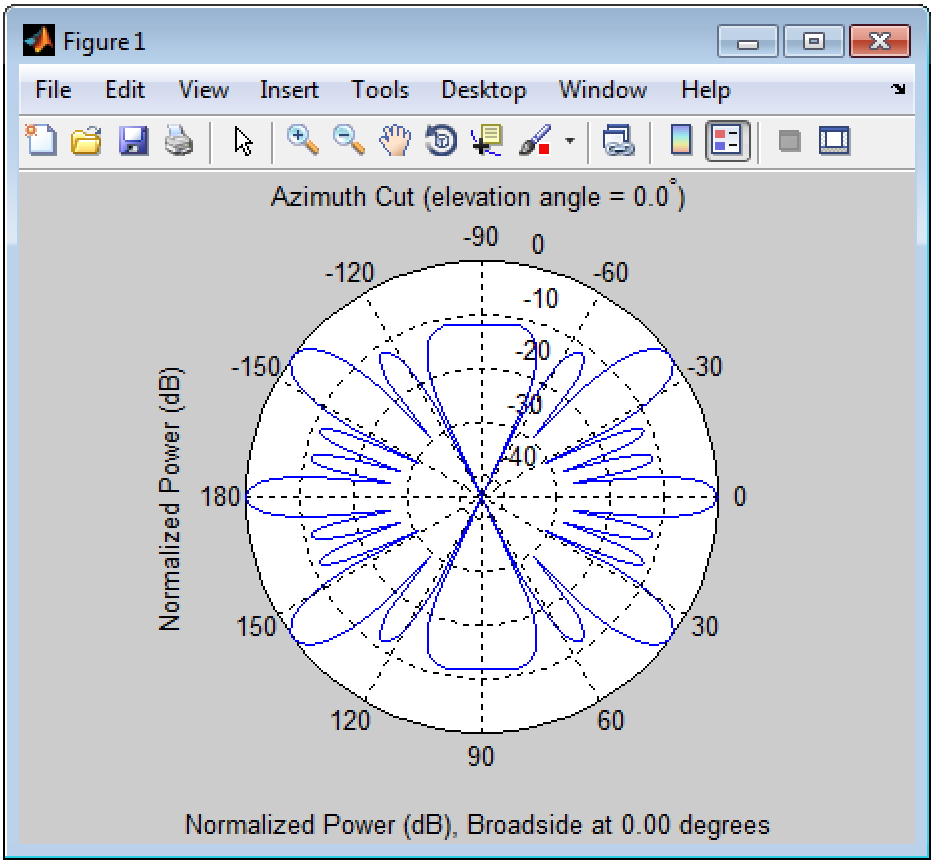
Plot the azimuth response of a 4-element ULA partitioned into two 2-element ULAs.

Create a 4-element ULA, and partition it into 2-element ULAs.

```
h = phased.ULA('NumElements',4,'ElementSpacing',0.5);
ha = phased.PartitionedArray('Array',h,...
   '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 3e8 m/s.

```
plotResponse(ha,1e9,3e8,'RespCut','Az','Format','Polar');
```



See Also `uv2azel` | `azel2uv`

phased.PartitionedArray.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.PartitionedArray.step

Purpose

Output responses of subarrays

Syntax

```
RESP = step(H,FREQ,ANG,V)
RESP = step(H,FREQ,ANG,V,STEERANGLE)
```

Description

`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 subarray's steering direction. This syntax is available when you set the SubarraySteering property to either `'Phase'` or `'Time'`.

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

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.

Output Arguments

RESP

Responses of subarrays of array. `RESP` has dimensions N-by-M-by-L. N is the number of subarrays in the phased array. 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`.

Examples

Response of Subarrays in Partitioned ULA

Calculate the response at the boresight of a 4-element ULA partitioned into two 2-element ULAs.

Create a 4-element ULA, and partition it into 2-element ULAs.

```
h = phased.ULA('NumElements',4,'ElementSpacing',0.5);
ha = phased.PartitionedArray('Array',h,...
    '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 3e8 m/s.

```
RESP = step(ha,1e9,[0;0],3e8);
```

See Also

`uv2azel` | `phitheta2azel`

Purpose

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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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 string `'All'` to show indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

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`

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 string `'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

String specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

Handles of array elements in figure window.

Examples

Plots Highlighting Overlapped Subarrays

Display the geometry of a uniform linear array having overlapped subarrays.

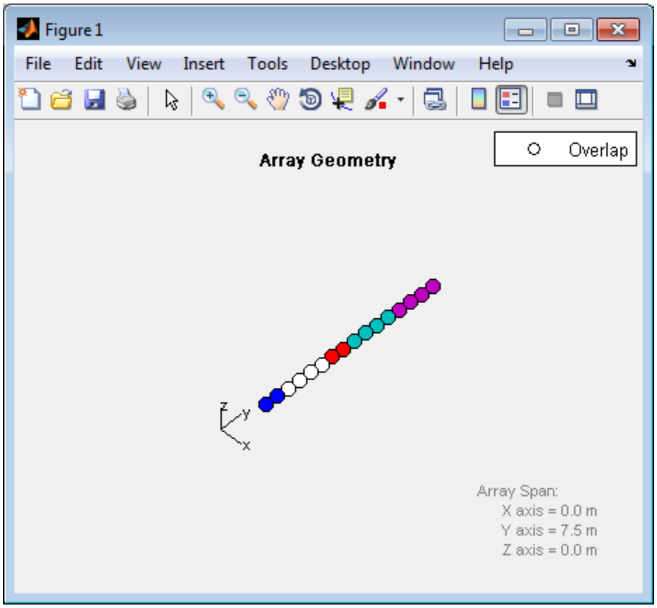
Create a 16-element ULA that has five 4-element subarrays. Some elements occur in more than one subarray.

```
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);
```

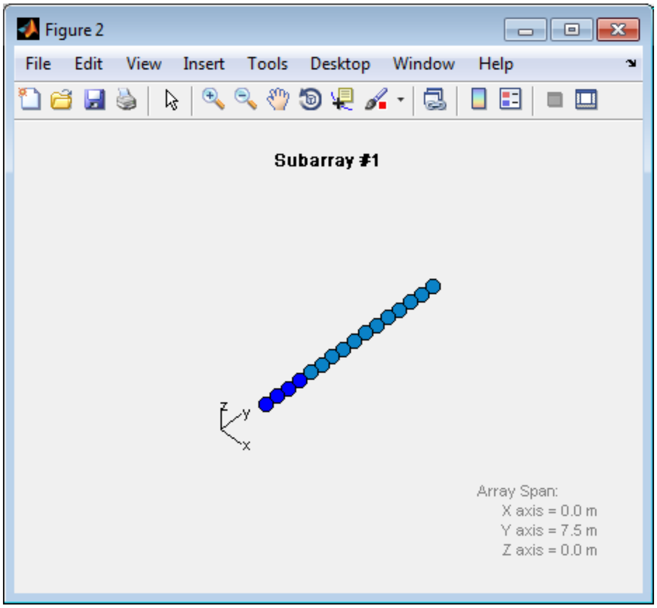


Each color other than white represents a different subarray. White represents elements that occur in multiple subarrays.

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
```



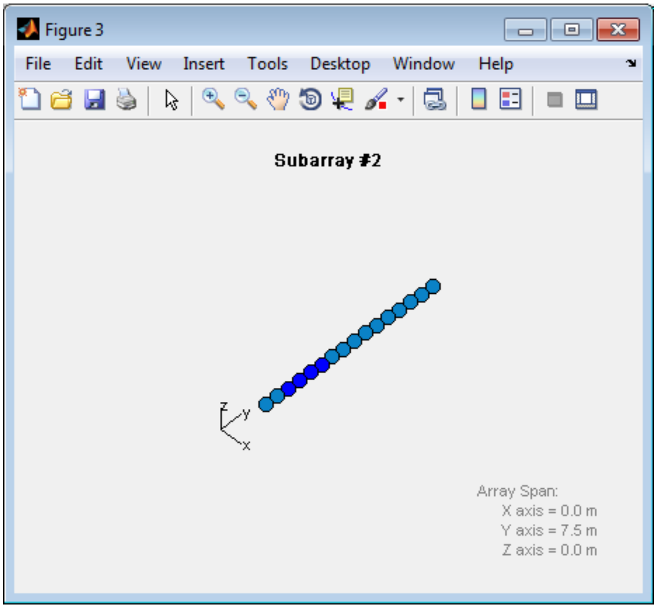
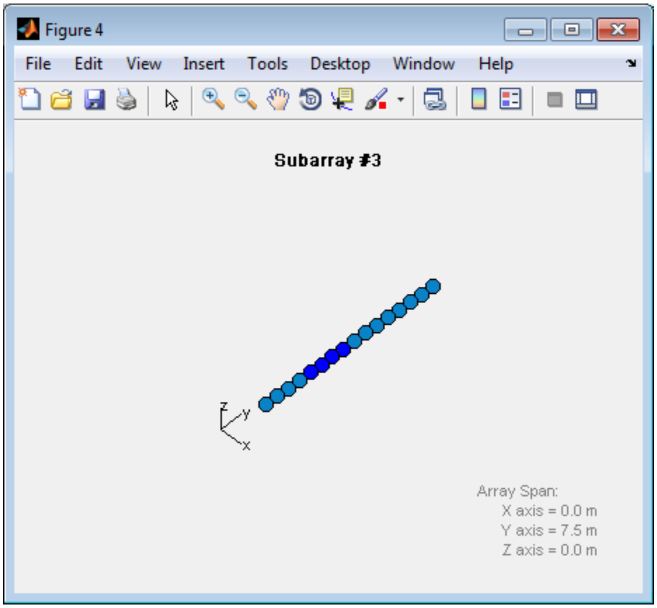


Figure 3
File Edit View Insert Tools Desktop Window Help
Subarray #2
z
y
x
Array Span:
X axis = 0.0 m
Y axis = 7.5 m
Z axis = 0.0 m



See Also `phased.ArrayResponse` |

Related Examples

- Phased Array Gallery

Purpose

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 3-545.
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.

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, 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.
- `(SampleRate * ChipWidth)` is an integer value.

Default: `1e6`

Type

Type of phase code

Specify the type of code used in phase modulation. Valid values are:

- `'Barker'`
- `'Frank'`
- `'P1'`
- `'P2'`
- `'P3'`
- `'P4'`
- `'Px'`
- `'Zadoff-Chu'`

Default: `'Frank'`

ChipWidth

Duration of each chip

Specify the duration of each chip in a phase-coded waveform in seconds as a positive scalar.

The value of this property must satisfy these constraints:

- `ChipWidth` is less than or equal to `(1./(NumChips * PRF))`.
- `(SampleRate * ChipWidth)` is an integer value.

Default: `1e-5`

NumChips

Number of chips

Specify the number of chips 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 table shows additional constraints on the number of chips for different code types.

If Type Property Is...	**Then 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 `Type` 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

Specify the pulse repetition frequency (in hertz) as a scalar or a row vector. The default value of this property corresponds to 10 kHz.

To implement a constant PRF, specify `PRF` as a positive scalar. To implement a staggered PRF, specify `PRF` as a row vector with positive elements. When `PRF` is a vector, the output pulses use successive elements of the vector as the PRF. If the last element of the vector is reached, the process continues cyclically with the first element of the vector.

The value of this property must satisfy these constraints:

- `PRF` is less than or equal to `(1/PulseWidth)`.
- `(SampleRate ./ PRF)` is a scalar or vector that contains only integers.

Default: `1e4`

OutputFormat

Output signal format

Specify the format of the output signal as one of `'Pulses'` or `'Samples'`. When you set the `OutputFormat` property to `'Pulses'`, the output of the `step` method is in the form of multiple pulses. In this case, the number of pulses is the value of the `NumPulses` property.

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: `'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`

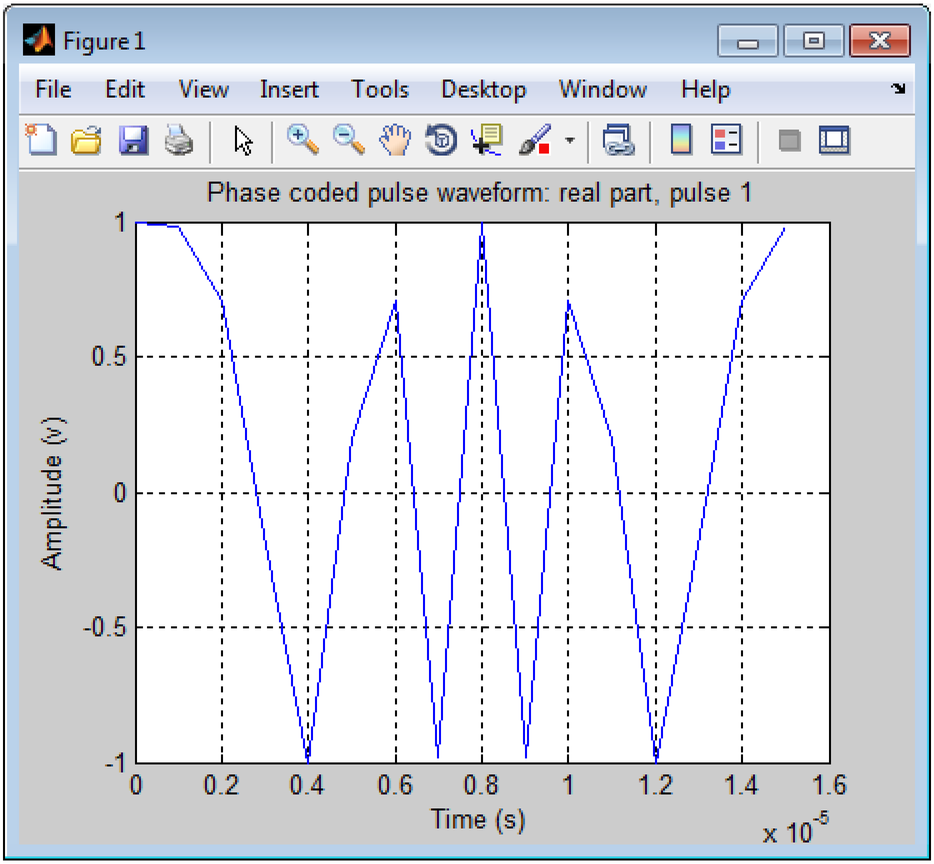
Methods

bandwidth	Bandwidth of phase-coded waveform
clone	Create phase-coded waveform object with same property values
getMatchedFilter	Matched filter coefficients for waveform
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plot	Plot phase-coded pulse waveform
release	Allow property value and input characteristics changes
reset	Reset states of phase-coded waveform object
step	Samples of phase-coded waveform

Examples

Create and plot a phase-coded pulse waveform that uses the Zadoff-Chu code.

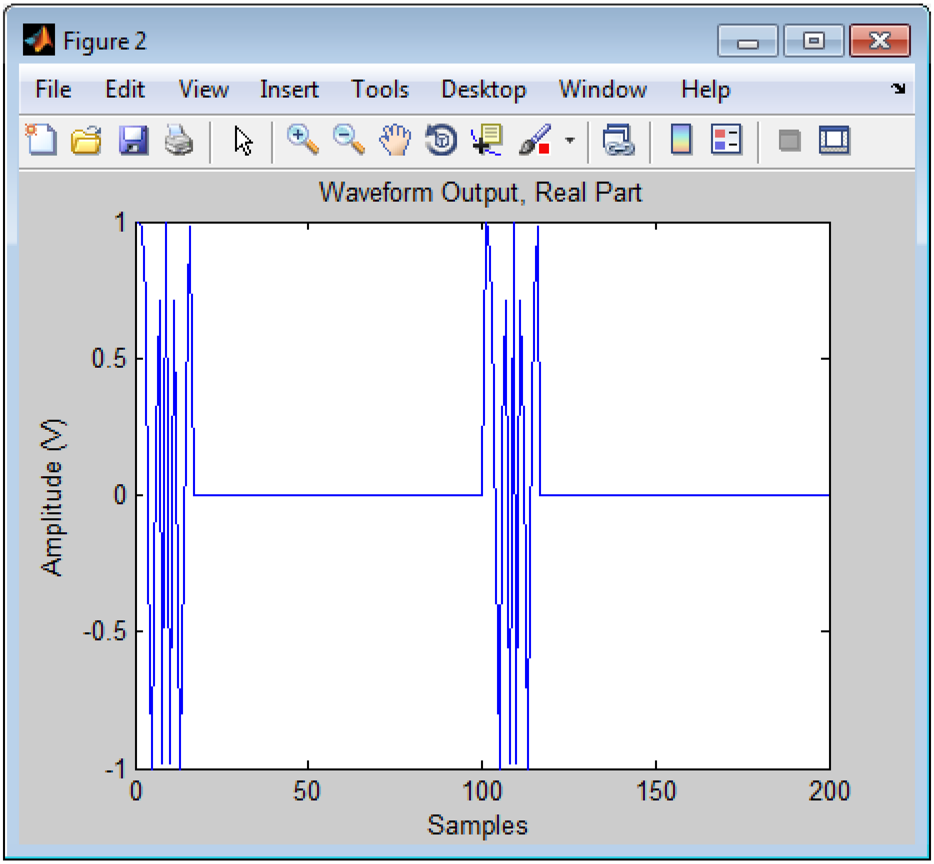
```
hw = phased.PhaseCodedWaveform('Type','Zadoff-Chu',...
    'ChipWidth',1e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
plot(hw);
```



Generate samples of a phase-coded pulse waveform that uses the Zadoff-Chu code, and plot the samples.

```
hw = phased.PhaseCodedWaveform('Type','Zadoff-Chu',...
```

```
    'ChipWidth',1e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
x = step(hw);
figure;
plot(real(x)); title('Waveform Output, Real Part');
xlabel('Samples'); ylabel('Amplitude (V)');
```



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 $\pi \cdot f(k) \cdot$ `SequenceIndex/NumChips` instead of $-\pi \cdot f(k) \cdot$ `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].

References

[1] Levanon, N. and E. Mozeson. *Radar Signals*. Hoboken, NJ: John Wiley & Sons, 2004.

See Also

`phased.LinearFMWaveform` **|** `phased.SteppedFMWaveform` **|** `phased.RectangularWaveform` **|**

Related Examples

- Waveform Analysis Using the Ambiguity Function

Concepts

- “Phase-Coded Waveforms”

phased.PhaseCodedWaveform.bandwidth

Purpose

Bandwidth of phase-coded waveform

Syntax

```
BW = bandwidth(H)
```

Description

`BW = bandwidth(H)` returns the bandwidth (in hertz) of the pulses for the phase-coded pulse waveform, `H`. The bandwidth value is the reciprocal of the chip width.

Input Arguments

H

Phase-coded waveform object.

Output Arguments

BW

Bandwidth of the pulses, in hertz.

Examples

Determine the bandwidth of a Frank code waveform.

```
H = phased.PhaseCodedWaveform;
bw = bandwidth(H);
```

phased.PhaseCodedWaveform.clone

Purpose Create phase-coded waveform object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.PhaseCodedWaveform.getMatchedFilter

Purpose

Matched filter coefficients for waveform

Syntax

`Coeff = getMatchedFilter(H)`

Description

`Coeff = getMatchedFilter(H)` returns the matched filter coefficients for the phase-coded waveform object, `H`. `Coeff` is a column vector.

Input Arguments

H

Phase-coded waveform object.

Output Arguments

Coeff

Column vector containing coefficients of the matched filter for `H`.

Examples

Get the matched filter coefficients for a phase-coded pulse waveform that uses the Zadoff-Chu code.

```
hwav = phased.PhaseCodedWaveform('Type','Zadoff-Chu',...
    'ChipWidth',1e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
coeff = getMatchedFilter(hwav);
stem(real(coeff));
title('Matched Filter Coefficients, Real Part');
axis([0 17 -1.1 1.1])
```

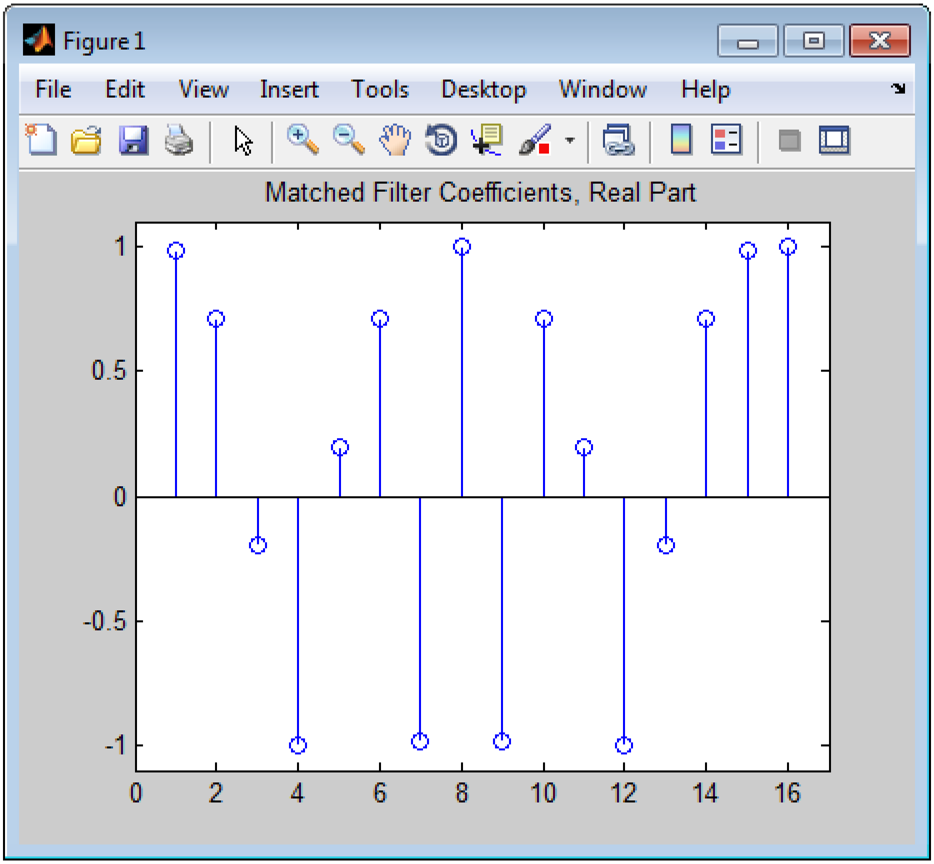


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Matched Filter Coefficients, Real Part
1
0.5
0
-0.5
-1
0
2
4
6
8
10
12
14
16

phased.PhaseCodedWaveform.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

```
N = getNumInputs(H)
```

Description

`N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.PhaseCodedWaveform.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `PhaseCodedWaveform` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.PhaseCodedWaveform.plot

Purpose 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

String that specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `Type` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Default: `'b'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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`

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

Create and plot a phase-coded pulse waveform that uses the Zadoff-Chu code.

```
hw = phased.PhaseCodedWaveform('Type','Zadoff-Chu',...
    'ChipWidth',1e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
plot(hw);
```

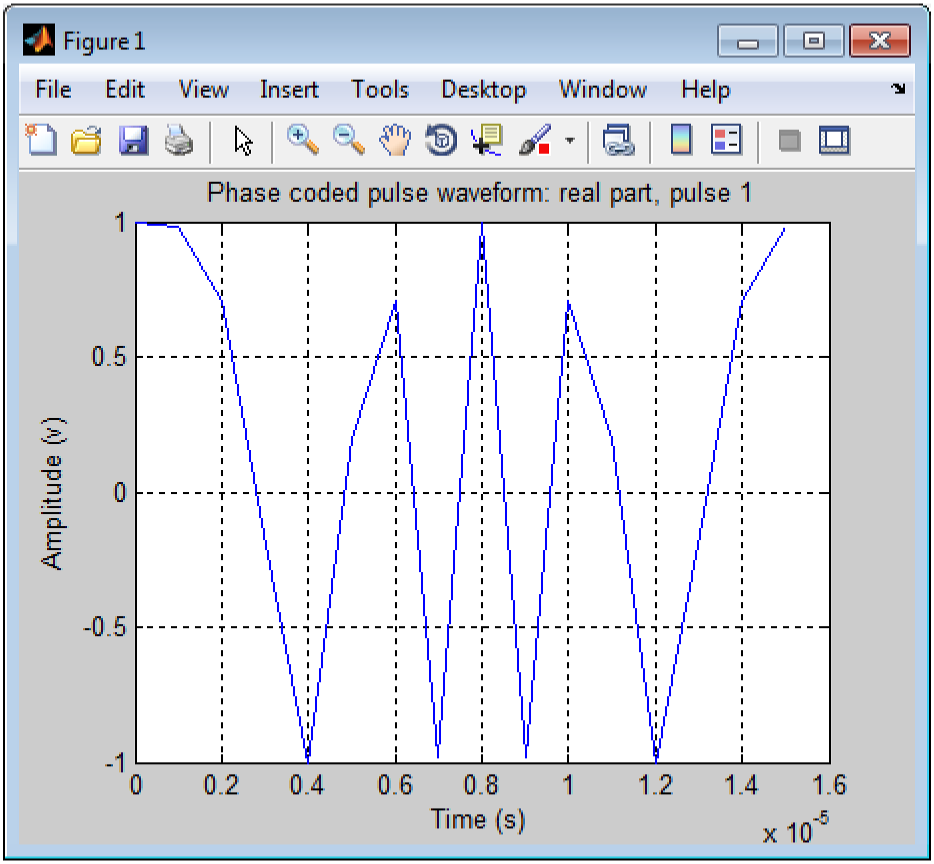


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Phase coded pulse waveform: real part, pulse 1
Amplitude (v)
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
Time (s)
x 10^-5

phased.PhaseCodedWaveform.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.PhaseCodedWaveform.reset

Purpose 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.

Purpose

Samples of phase-coded waveform

Syntax

`Y = step(H)`

Description

`Y = step(H)` returns samples of the phase-coded pulse in a column vector, `Y`.

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

H

Phase-coded waveform object.

Output Arguments

Y

Column vector containing the waveform samples.

Examples

Generate samples of two pulses of a phase-coded pulse waveform that uses the Zadoff-Chu code.

```
hw = phased.PhaseCodedWaveform('Type','Zadoff-Chu',...
    'ChipWidth',1e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
x = step(hw);
figure;
plot(real(x)); title('Waveform Output, Real Part');
xlabel('Samples'); ylabel('Amplitude (V)');
```

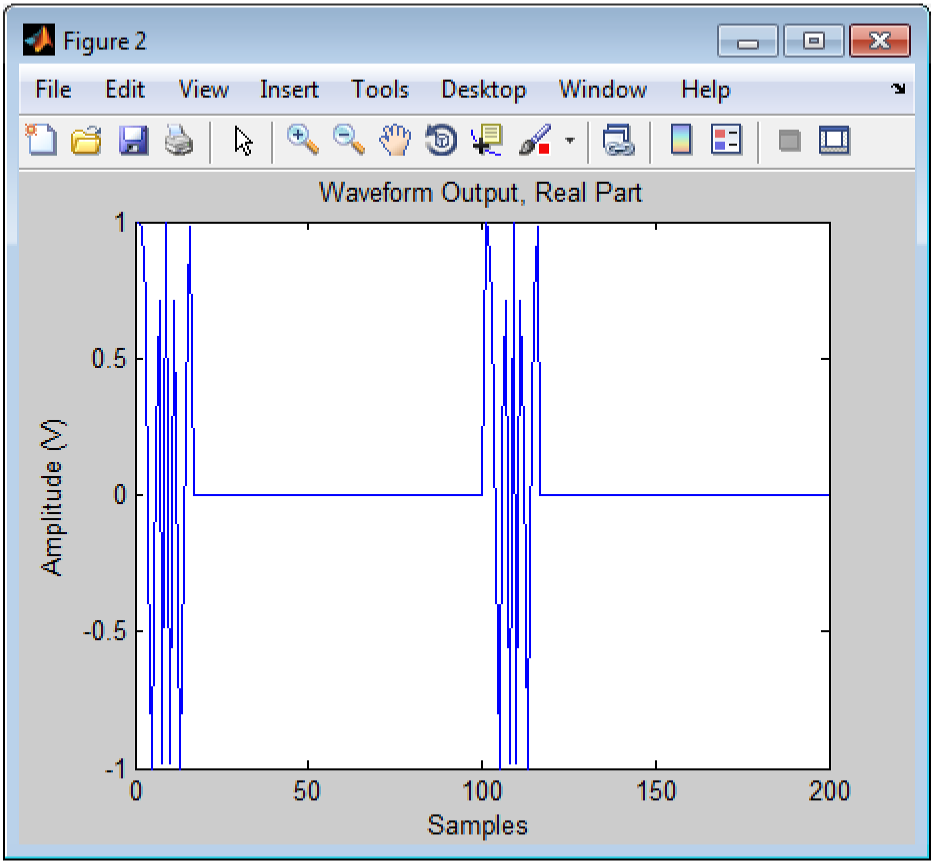


Figure 2
File
Edit
View
Insert
Tools
Desktop
Window
Help
Waveform Output, Real Part
Amplitude (V)
Samples
1
0.5
0
-0.5
-1
0
50
100
150
200

Purpose

Narrowband phase shift beamformer

Description

The `PhaseShiftBeamformer` object implements a phase shift beamformer.

To compute the beamformed signal:

1. Define and set up your phase shift beamformer. See "Construction" on page 3-567.
2. Call `step` to perform the beamforming operation according to the properties of `phased.PhaseShiftBeamformer`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.PhaseShiftBeamformer` creates a conventional phase shift beamformer System object, `H`. The object performs phase shift beamforming on the received signal.

`H = phased.PhaseShiftBeamformer(Name,Value)` creates a 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

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 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

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.

Default: `3e8`

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'`.

Default: `[0; 0]`

WeightsNormalization

Approach for normalizing beamformer weights

If you set this property value to `'Distortionless'`, the gain toward the beamforming direction is 0 dB. If you set this property value to `'Preserve power'`, the norm of the weights is 1.

Default: `'Distortionless'`

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

clone	Create phase shift beamformer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform phase shift beamforming

Examples

Apply phase shift beamforming to the signal received by a 5-element ULA. The beamforming direction is 45 degrees azimuth and 0 degrees elevation.

```
% Simulate signal
t = (0:1000)';
x = sin(2*pi*0.01*t);
c = 3e8; Fc = 3e8;
incidentAngle = [45; 0];
ha = phased.ULA('NumElements',5);
x = collectPlaneWave(ha,x,incidentAngle,Fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;

% Beamforming
hbf = phased.PhaseShiftBeamformer('SensorArray',ha,...
    'OperatingFrequency',Fc,'PropagationSpeed',c,...
    'Direction',incidentAngle,'WeightsOutputPort',true);
[y,w] = step(hbf,rx);

% Plot signals
plot(t,real(rx(:,3)),'r:',t,real(y));
xlabel('Time'); ylabel('Amplitude');
legend('Original','Beamformed');

% Plot response pattern
figure;
plotResponse(ha,Fc,c,'Weights',w);
```

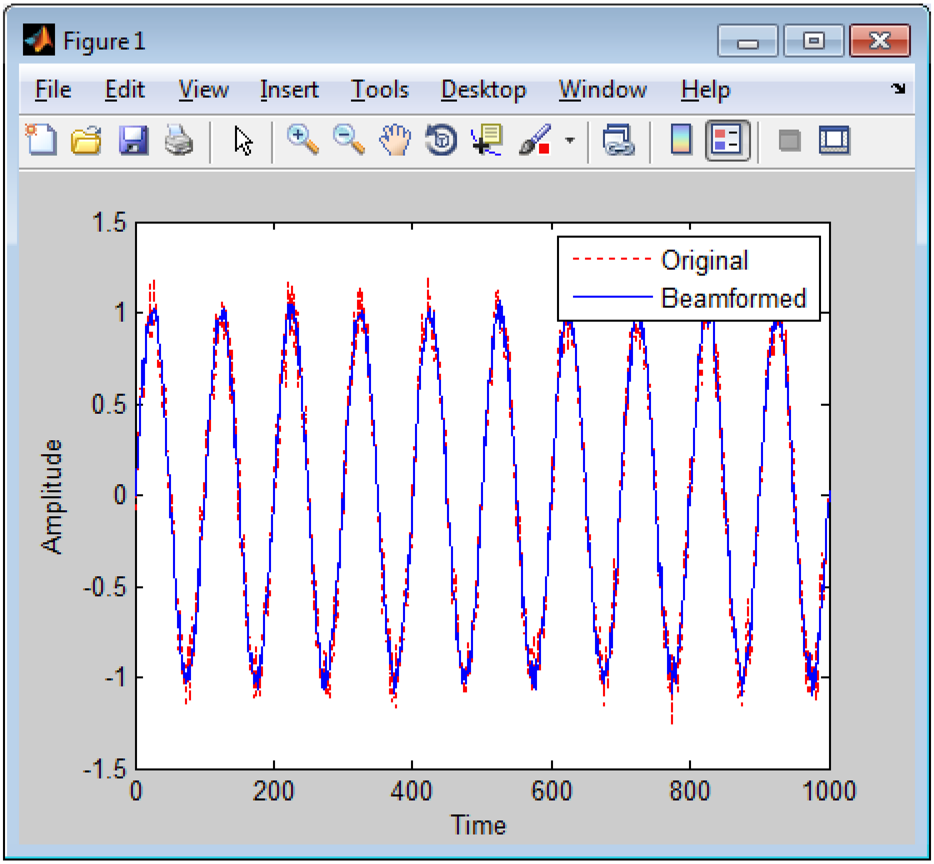
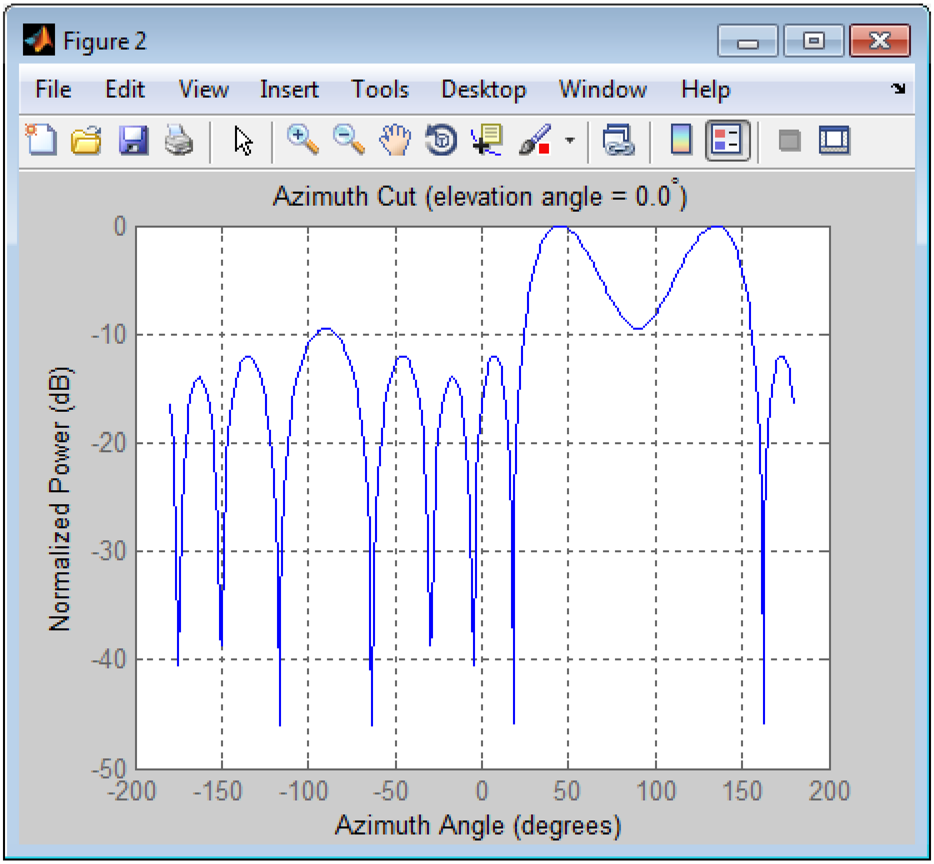


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Original
Beamformed
Amplitude
Time
1.5
1
0.5
0
-0.5
-1
-1.5
0
200
400
600
800
1000



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

`phased.LCMVBeamformer` | `phased.MVDRBeamformer` | `phased.SubbandPhaseShiftBeamformer` | `uv2azel` | `phitheta2azel`

phased.PhaseShiftBeamformer.clone

Purpose Create phase shift beamformer object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax

```
N = getNumInputs(H)
```

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.PhaseShiftBeamformer.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.PhaseShiftBeamformer.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `PhaseShiftBeamformer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.PhaseShiftBeamformer.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform phase shift beamforming

Syntax

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

Description

`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 `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

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.

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

Apply phase shift beamforming to the signal received by a 5-element ULA. The beamforming direction is 45 degrees azimuth and 0 degrees elevation.

```
% Simulate signal
t = (0:1000)';
x = sin(2*pi*0.01*t);
c = 3e8; Fc = 3e8;
incidentAngle = [45; 0];
ha = phased.ULA('NumElements',5);
x = collectPlaneWave(ha,x,incidentAngle,Fc,c);
noise = 0.1*(randn(size(x)) + 1j*randn(size(x)));
rx = x + noise;

% Beamforming
hbf = phased.PhaseShiftBeamformer('SensorArray',ha,...
    'OperatingFrequency',Fc,'PropagationSpeed',c,...
    'Direction',incidentAngle,'WeightsOutputPort',true);
[y,w] = step(hbf,rx);
```

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`

Purpose

Motion platform

Description

The `Platform` object models the translational motion of a target or array in space.

To model a moving platform:

1. Define and set up your platform. See "Construction" on page 3-582.
2. Call `step` to move the platform following a defined path according to the properties of `phased.Platform`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.Platform` creates a platform System object, `H`. The object models translational motion in space.

`H = phased.Platform(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.Platform(POS,V,Name,Value))` creates a platform object, `H`, with the `InitialPosition` property set to `POS`, the `Velocity` property set to `V`, and other specified property Names set to the specified Values. `POS` and `V` are value-only arguments. To specify a value-only argument, you must also specify all preceding value-only arguments. You can specify name-value pair arguments in any order.

Properties

InitialPosition

Initial position of platform

Specify the initial position of the platform as a 3-by-1 column vector in the form of `[x; y; z]` (in meters).

Default: `[0; 0; 0]`

Velocity

Velocity of platform

Specify the current velocity of the platform as a 3-by-1 vector in the form of `[x; y; z]` (in meters/second). This property is tunable.

Default: `[0; 0; 0]`

OrientationAxes

Orientation axes of platform

Specify the three axes that define the local (x, y, z) coordinate system at the platform as a 3-by-3 matrix (one axis in each column). The three axes must be orthonormal.

Default: `[1 0 0;0 1 0;0 0 1]`

OrientationAxesOutputPort

Output orientation axes

To obtain the 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

clone	Create platform object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset platform to initial position
step	Output current position, velocity, and orientation axes of platform

Examples

Define a platform at origin with a velocity of (100,100,0) in meters per second. Simulate the motion of the platform for 2 steps, assuming the time elapsed for each step is 1 second.

```
Hp = phased.Platform([0; 0; 0],[100; 100; 0]);
T = 1;
[pos,v] = step(Hp,T)
[pos,v] = step(Hp,T)
```

See Also

`global2localcoord` | `local2globalcoordphased.Collector` **|** `phased.Radiator` **|** | `rangeangle`

Related Examples

- “Motion Modeling in Phased Array Systems”

Purpose Create platform object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.Platform.getNumInputs

Purpose Number of expected inputs to step method

Syntax

```
N = getNumInputs(H)
```

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose

Number of outputs from step method

Syntax

```
N = getNumOutputs(H)
```

Description

`N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.Platform.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `Platform` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.Platform.release

Purpose

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.Platform.reset

Purpose Reset platform to initial position

Syntax `reset(H)`

Description `reset(H)` resets the initial position of the `Platform` object, `H`.

Purpose

Output current position, velocity, and orientation axes of platform

Syntax

```
[P,V] = step(H,T)
[P,V,AX] = step(H,T)
```

Description

`[P,V] = step(H,T)` returns the current position, `P`, and the current velocity, `V`, of the platform. The method then updates the position and velocity using the equation P = P+VT where `T` specifies the elapsed time (in seconds) for the current step.

`[P,V,AX] = step(H,T)` returns the additional output `AX` as the platform's orientation axes when you set the `OrientationAxesOutputPort` property to `true`.

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.

Examples

Define a platform at origin with a velocity of [100; 100; 0] in meters per second. Simulate the motion of the platform for 2 steps, assuming the time elapsed for each step is 1 second.

```
Hp = phased.Platform([0; 0; 0],[100; 100; 0]);
T = 1;
[pos,v] = step(Hp,T)
[pos,v] = step(Hp,T)
```

Purpose

Radar target

Description

The `RadarTarget` object models a radar target.

To compute the signal reflected from a radar target:

1. Define and set up your radar target. See "Construction" on page 3-592.
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.

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

MeanRCSSource

Source of mean radar cross section

Specify whether the target's mean RCS value 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.
`'Input port'`	An input argument in each invocation of `step` specifies the mean RCS value.

Default: `'Property'`

MeanRCS

Mean radar cross section

Specify the mean value of the target's radar cross section (in square meters) as a nonnegative scalar. This property applies when the `MeanRCSSource` property is `'Property'`. This property is tunable.

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`.

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. The default value of this property corresponds to 300 MHz.

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

clone	Create radar target object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset states of radar target object
step	Reflect incoming signal

Examples

Calculate the reflected signal from a nonfluctuating point target.

```
x = ones(10,1);
hr = phased.RadarTarget('Model','Nonfluctuating','MeanRCS',10);
y = step(hr,x);
```

Algorithms

The reflected signal is given by:

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

where:

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

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

- σ is the mean RCS of the target

- λ is the wavelength of the incoming signal

Each element of the signal incident on the target is scaled by the gain factor.

For further details, see [1].

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

`phased.FreeSpace` | `phased.Platform` |

Concepts

- “Radar Target”

Purpose

Create radar target object with same property values

Syntax

`C = clone(H)`

Description

`C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.RadarTarget.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.RadarTarget.getNumOutputs

Purpose Number of outputs from step method

Syntax

```
N = getNumOutputs(H)
```

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.RadarTarget.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF` of the `RadarTarget` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles, or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.RadarTarget.reset

Purpose 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'`.

Purpose

Reflect incoming signal

Syntax

```
Y = step(H,X)
Y = step(H,X,MEANRCS)
Y = step(H,X,UPDATERCS)
Y = step(H,X,MEANRCS,UPDATERCS)
```

Description

`Y = step(H,X)` returns the reflected signal `Y` due to the incident signal `X`. 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 RCS value.

`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'`. `MEANRCS` must be a positive scalar.

`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'`, `'Swerling 2'`, `'Swerling 3'`, or `'Swerling 4'`. If `UPDATERCS` is `true`, a new RCS value is generated. If `UPDATERCS` is `false`, the previous RCS value is used.

You can combine optional input arguments when their enabling properties are set: `Y = step(H,X,MEANRCS,UPDATERCS)`

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.

Examples

Reflect a 250-Hz sine wave with unit amplitude off a target with a nonfluctuating RCS of 2 square meters. The carrier frequency of the sine wave is 1 GHz.

```
htarget = phased.RadarTarget('Model','nonfluctuating',...
    'MeanRCS',2,'OperatingFrequency',1e9);
t = linspace(0,1,1000);
sig = cos(2*pi*250*t)';
reflectedsig = step(htarget,sig);
```

Algorithms

The reflected signal is given by:

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

where:

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

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

- σ is the mean RCS of the target
- λ is the wavelength of the incoming signal

Each element of the signal incident on the target is scaled by the gain factor.

For further details, see [1].

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.

Purpose

Narrowband signal radiator

Description

The `Radiator` object implements a narrowband signal radiator.

To compute the radiated signal from the sensor(s):

1. Define and set up your radiator. See "Construction" on page 3-605.
2. Call `step` to compute the radiated signal according to the properties of `phased.Radiator`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.Radiator` creates a narrowband signal radiator System object, `H`. The object returns radiated narrowband signals for given directions using a sensor array or a single element.

`H = phased.Radiator(Name,Value)` creates a radiator 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

Sensor

Handle of sensor

Specify the sensor as a sensor array object or an element object in the `phased` package. If the sensor is an array, it 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

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`

WeightsInputPort

Enable weights input

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`

CombineRadiatedSignals

Combine radiated signals

Set this property to `true` to combine radiated signals from all radiating elements. Set this property to `false` to obtain the radiated signal for each radiating element. If the `Sensor` property is an array that contains subarrays, the `CombineRadiatedSignals` property must be `true`.

Default: `true`

Methods

clone	Create radiator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Radiate signals

Examples

Radiate signal with a single antenna.

```
ha = phased.IsotropicAntennaElement;
hr = phased.Radiator('Sensor',ha,'OperatingFrequency',300e6);
x = [1;1];
radiatingAngle = [30 10]';
y = step(hr,x,radiatingAngle);
```

Radiate a far field signal with a 5-element array.

```
ha = phased.ULA('NumElements',5);
hr = phased.Radiator('Sensor',ha,'OperatingFrequency',300e6);
x = [1;1];
radiatingAngle = [30 10; 20 0]'; % two directions
y = step(hr,x,radiatingAngle);
```

Radiate signal with a 3-element antenna array. Each antenna radiates a separate signal to a separate direction.

```
ha = phased.ULA('NumElements',3);
hr = phased.Radiator('Sensor',ha,'OperatingFrequency',1e9,...
    'CombineRadiatedSignals',false);
x = [1 2 3;1 2 3];
radiatingAngle = [10 0; 20 5; 45 2]'; % One angle for one antenna
y = step(hr,x,radiatingAngle);
```

References

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

See Also

`phased.Collector` |

Purpose Create radiator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.Radiator.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.Radiator.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `Radiator` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.Radiator.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

phased.Radiator.step

Purpose Radiate signals

Syntax

```
Y = step(H,X,ANG)
Y = step(H,X,ANG,WEIGHTS)
Y = step(H,X,ANG,STEERANGLE)
Y = step(H,X,ANG,WEIGHTS,STEERANGLE)
```

Description

`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,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,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 `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

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.

ANG

Incident directions of signals. `ANG` is a two-row matrix. Each column specifies a radiating direction in the form [AzimuthAngle; ElevationAngle], in degrees.

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 degrees, inclusive.

Output Arguments

`Y`

Radiated signals. `Y` is a matrix whose number of columns equals the number of radiating directions in `ANG`. Each column of `Y` contains the output from all radiating elements or subarrays. The output is the result of radiating the signal in all directions in `ANG`, or one direction in `ANG`, depending on the `CombineRadiatedSignals` property of `H`.

Examples

Radiate a far field signal with a 5-element uniform linear array.

```
ha = phased.ULA('NumElements',5);
% construct the radiator object
hr = phased.Radiator('Sensor',ha,...
    'OperatingFrequency',300e6,'CombineRadiatedSignals',true);
% simple signal to radiate
x = [1;1];
% radiating direction in azimuth and elevation
radiatingAngle = [30; 10];
% use the step method to radiate the signal
y = step(hr,x,radiatingAngle);
```

phased.RangeDopplerResponse

Purpose

Range-Doppler response

Description

The `RangeDopplerResponse` object calculates the range-Doppler response of input data.

To compute the range-Doppler response:

1. Define and set up your range-Doppler response calculator. See "Construction" on page 3-617.
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.

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, 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

Method of range processing

Specify the method of range processing as `'Matched filter'` or `'Dechirp'`.

`'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.
`'Dechirp'`	Algorithm mixes the incoming signal with a reference signal. This approach is common with FMCW signals, where the reference signal is the transmitted signal. This approach can also apply to a system that uses 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.

Default: Speed of light

SampleRate

Sample rate

Specify the sample rate, in hertz, as a positive scalar. 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 applies only when you set the `RangeMethod` property to `'Dechirp'`.

Default: `1e9`

DechirpInput

Whether 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. This property applies only when you set the `RangeMethod` property to `'Dechirp'`.

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.

This property applies only when you set the `RangeMethod` property to `'Dechirp'` and the `DechirpInput` property to `true`. The default value indicates no decimation.

Default: `1`

RangeFFTLengthSource

Source of FFT length in range processing

Specify how the object determines the FFT length 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.

This property applies only when you set the `RangeMethod` property to `'Dechirp'`.

Default: `'Auto'`

RangeFFTLength

FFT length in range processing

Specify the FFT length in the range domain as a positive integer. This property applies only when you set the `RangeMethod` property to `'Dechirp'` 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. This property applies only when you set the `RangeMethod` property to `'Dechirp'`.

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 applies only when you set the `RangeMethod` property to `'Dechirp'` 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. This property applies only when you set the `RangeMethod` property to `'Dechirp'` 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`

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.

This property applies only when you set the `RangeMethod` property to `'Dechirp'`.

Default: `'Auto'`

DopplerFFTLength

FFT length in Doppler processing

Specify the FFT length in Doppler processing as a positive integer. This property applies only when you set the `RangeMethod` property to `'Dechirp'` 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. This property applies only when you set the `RangeMethod` property to `'Dechirp'`.

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 applies only when you set the `RangeMethod` property to `'Dechirp'` 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. This property applies only when you set the `RangeMethod` property to `'Dechirp'` and the `DopplerWindow` property to `'Custom'`.

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.

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. This property applies only when you set the `DopplerOutput` property to `'Speed'`. The default value of this property corresponds to 300 MHz.

Default: `3e8`

Methods

clone	Create range-Doppler response object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot range-Doppler response
release	Allow property value and input characteristics changes
step	Calculate range-Doppler response

Examples

Range-Doppler Response of Pulsed Radar Signal Using 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.

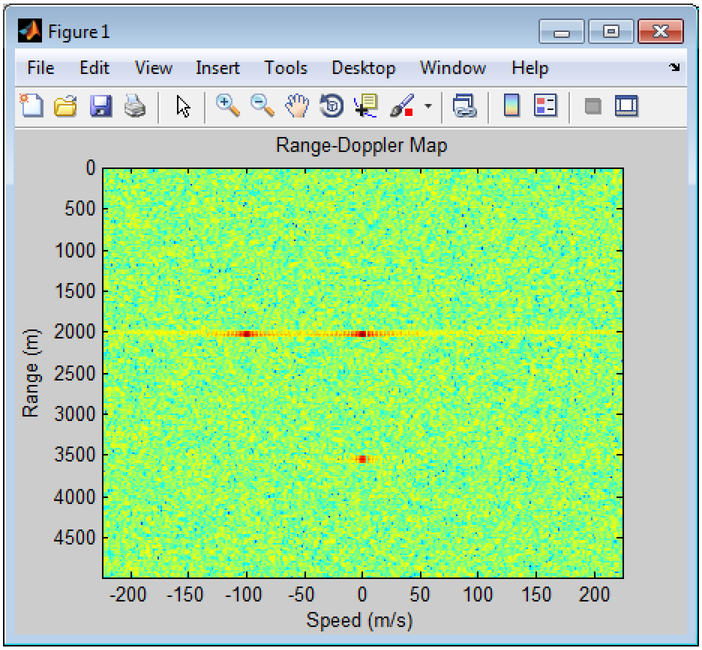
```
hrdresp = phased.RangeDopplerResponse(...
   'DopplerFFTLengthSource','Property',...
   'DopplerFFTLength',RangeDopplerEx_MF_NFFTDOP,...
   'SampleRate',RangeDopplerEx_MF_Fs,...
   'DopplerOutput','Speed',...
   'OperatingFrequency',RangeDopplerEx_MF_Fc);
```

Calculate the range-Doppler response.

```
[resp,rng_grid,dop_grid] = step(hrdresp,...
   RangeDopplerEx_MF_X,RangeDopplerEx_MF_Coeff);
```

Plot the range-Doppler map.

```
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

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 about –0.36 relative to the radar.

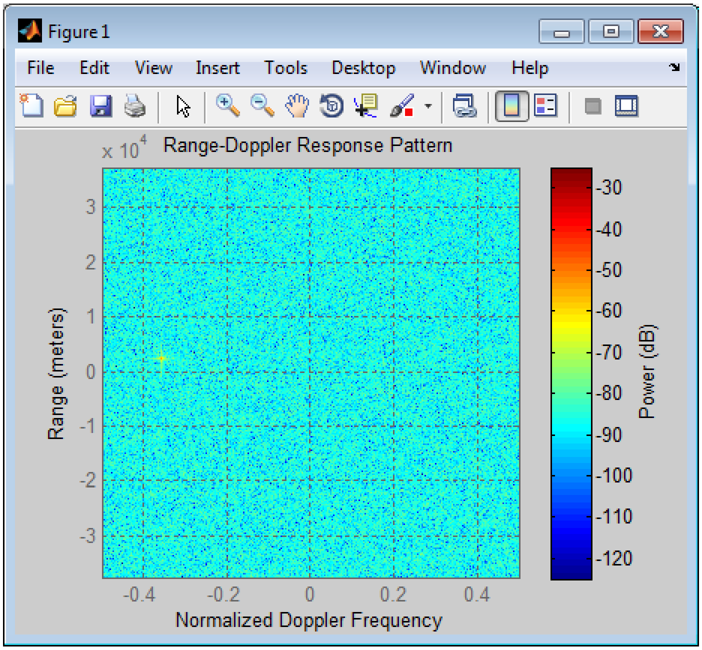
```
load RangeDopplerExampleData;
```

Create a range-Doppler response object.

```
hrdresp = phased.RangeDopplerResponse(...
   'RangeMethod','Dechirp',...
   '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)
```



Algorithms

The RangeDopplerResponse object generates the response as follows:

1. Processes the input signal in the range domain using either a matched filter or dechirp operation.
2. Processes in the Doppler domain 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.

See Also `phased.AngleDopplerResponse` | `phased.MatchedFilter` | `dechirp`

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

Purpose Create range-Doppler response object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.RangeDopplerResponse.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.RangeDopplerResponse.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.RangeDopplerResponse.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `RangeDopplerResponse` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.RangeDopplerResponse.plotResponse

Purpose

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 `'Dechirp'` 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 `'Dechirp'` 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.

xref

Reference signal, specified as a column vector having the same number of rows as `x`.

coeff

Matched filter coefficients, specified as a column vector.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

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 about –0.36 relative to the radar.

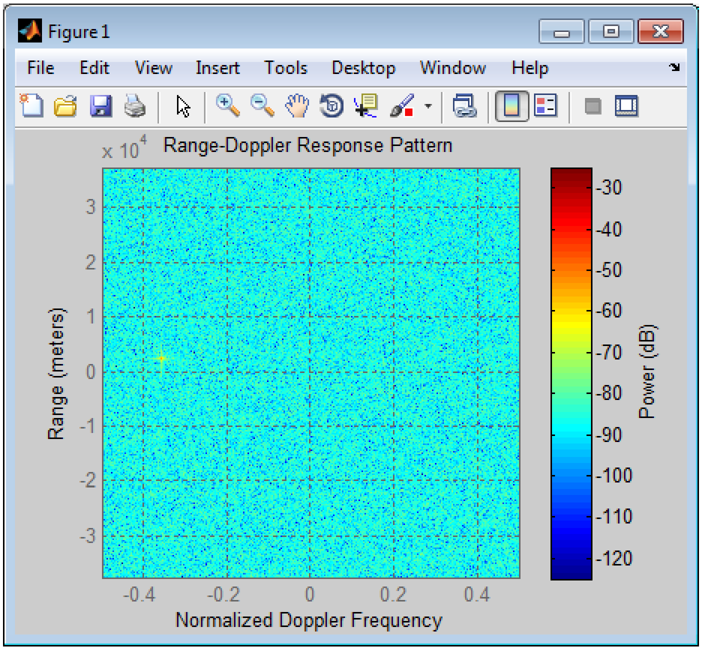
```
load RangeDopplerExampleData;
```

Create a range-Doppler response object.

```
hrdresp = phased.RangeDopplerResponse(...
   'RangeMethod','Dechirp',...
   '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

`phased.AngleDopplerResponse.plotResponse` |

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.RangeDopplerResponse.step

Purpose Calculate range-Doppler response

Syntax

```
[RESP,RNG_GRID,DOP_GRID] = step(H,x)
[RESP,RNG_GRID,DOP_GRID] = step(H,x,xref)
[RESP,RNG_GRID,DOP_GRID] = step(H,x,coeff)
```

Description

`[RESP,RNG_GRID,DOP_GRID] = step(H,x)` calculates the angle-Doppler response of the input signal, `x`. `RESP` is the complex range-Doppler response. `RNG_GRID` and `DOP_GRID` 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 `'Dechirp'` and the `DechirpInput` property to `false`. This syntax is most commonly used with FMCW signals.

`[RESP,RNG_GRID,DOP_GRID] = step(H,x,xref)` uses `xref` as the reference signal to dechirp `x`. This syntax is available when you set the `RangeMethod` property to `'Dechirp'` 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,RNG_GRID,DOP_GRID] = 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.

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

H

Range-Doppler response object.

x

Input data. Specific requirements depend on the syntax:

- In the syntax `step(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 `step(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 `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.

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.

xref

Reference signal, specified as a column vector having the same number of rows as `x`.

coeff

Matched filter coefficients, specified as a column vector.

Output Arguments

RESP

Complex range-Doppler response of `x`, returned as a P-by-Q matrix. The values of P and Q depend on the syntax.

Syntax	Values of P and Q
`step(H,x)`	If you set the `RangeFFTLength` property to `'Auto'`, P is the number of rows in `x`. Otherwise, P is the value of the `RangeFFTLength` property. If you set the `DopplerFFTLength` property to `'Auto'`, Q is the number of columns in `x`. Otherwise, Q is the value of the `DopplerFFTLength` property.
`step(H,x,xref)`	P is the quotient between the number of rows of `x` and the value of the `DecimationFactor` property. If you set the `DopplerFFTLength` property to `'Auto'`, Q is the number of columns in `x`. Otherwise, Q is the value of the `DopplerFFTLength` property.
`step(H,x,coeff)`	P is the number of rows of `x`. If you set the `DopplerFFTLength` property to `'Auto'`, Q is the number of

Syntax	Values of P and Q
	columns in `X`. Otherwise, Q is the value of the `DopplerFFTLength` property.

RNG_GRID

Range samples at which the range-Doppler response is evaluated. `RNG_GRID` is a column vector of length P.

DOP_GRID

Doppler samples or speed samples at which the range-Doppler response is evaluated. `DOP_GRID` is a column vector of length Q. Whether `DOP_GRID` contains Doppler or speed samples depends on the `DopplerOutput` property of `H`.

Examples

Range-Doppler Response of Pulsed Radar Signal Using 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.

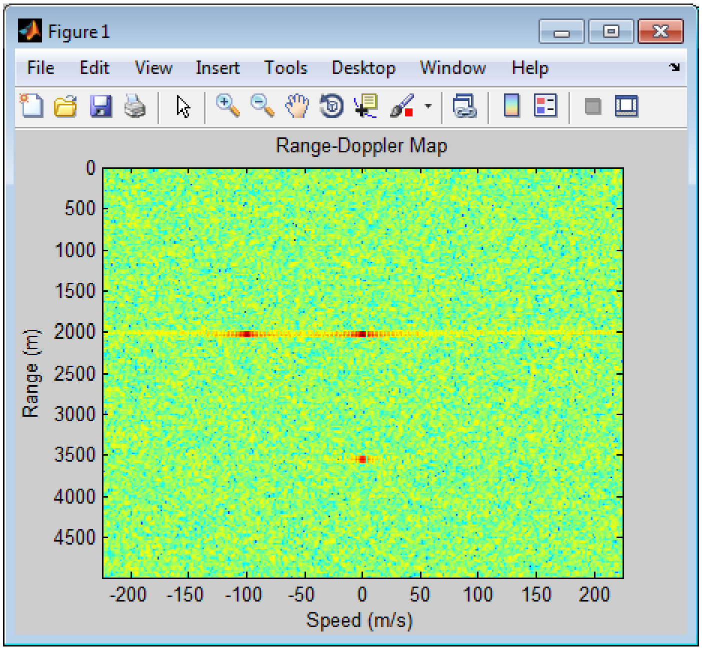
```
hrdresp = phased.RangeDopplerResponse(...
   'DopplerFFTLengthSource','Property',...
   'DopplerFFTLength',RangeDopplerEx_MF_NFFTDOP,...
   'SampleRate',RangeDopplerEx_MF_Fs,...
   'DopplerOutput','Speed',...
   'OperatingFrequency',RangeDopplerEx_MF_Fc);
```

Calculate the range-Doppler response.

```
[resp,rng_grid,dop_grid] = step(hrdresp,...
   RangeDopplerEx_MF_X,RangeDopplerEx_MF_Coeff);
```

Plot the range-Doppler map.

```
imagesc(dop_grid,rng_grid,mag2db(abs(resp)));
xlabel('Speed (m/s)');
ylabel('Range (m)');
title('Range-Doppler Map');
```



Estimation of Doppler and Range from Range-Doppler Response Data

Load data for an FMCW signal that has not been dechirped. The signal contains the return from one target.

```
load RangeDopplerExampleData;
```

Create a range-Doppler response object.

```
hrdresp = phased.RangeDopplerResponse(...
   'RangeMethod','Dechirp',...
   '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 based on the map.

```
[x_temp,idx_temp] = max(abs(resp));
[~,dop_idx] = max(x_temp);
rng_idx = idx_temp(dop_idx);
dop_est = dop_grid(dop_idx)
rng_est = rng_grid(rng_idx)
```

```
dop_est =

 -712.8906


rng_est =

        2250
```

The target is approximately 2250 m away, and it is moving fast enough to cause a Doppler shift of approximately –713 Hz.

Purpose

Receiver preamp

Description

The `ReceiverPreamp` object implements a receiver preamp.

To model a receiver preamp:

1. Define and set up your receiver preamp. See "Construction" on page 3-644.
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.

Construction

`H = phased.ReceiverPreamp` creates a receiver preamp System object, `H`. The object receives the incoming pulses.

`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`

NoiseBandwidth

Noise bandwidth of receiver

A scalar containing the bandwidth of noise spectrum (in hertz) at the receiver preamp. If the receiver has multiple channels/sensors, the noise bandwidth applies to each channel/sensor.

Default: `1e6`

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.

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.

Default: `290`

SampleRate

Sample rate

Specify the sample rate, in hertz, as a positive scalar. The default value corresponds to 1 MHz.

Default: `1e6`

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

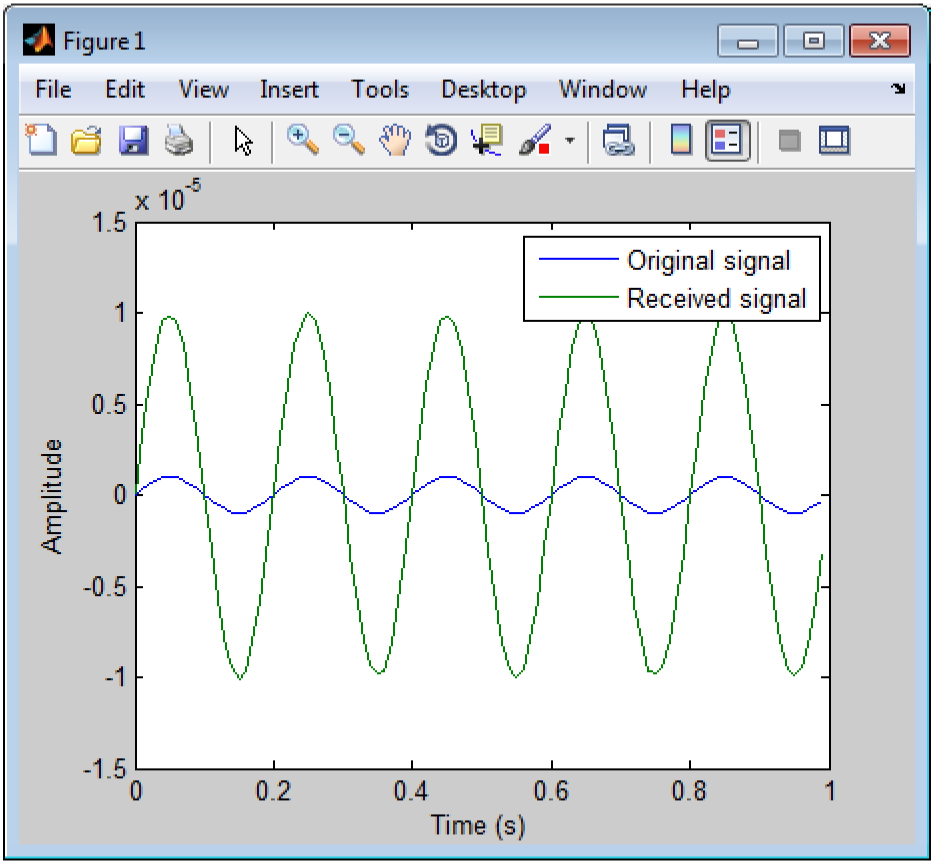
clone	Create receiver preamp object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset random number generator for noise generation
step	Receive incoming signal

Examples

Simulate the reception of a sine wave.

```
Hrx = phased.ReceiverPreamp('NoiseFigure',10);
Fs = 100;
t = linspace(0,1-1/Fs,100);
x = 1e-6*sin(2*pi*5*t);
y = step(Hrx,x);
plot(t,x,t,real(y));
```

```
xlabel('Time (s)'); ylabel('Amplitude');
legend('Original signal','Received signal');
```



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 `phased.Collector` | `phased.Transmitter` |

Concepts

- "Receiver Preamp"

phased.ReceiverPreamp.clone

Purpose Create receiver preamp object with same property values

Syntax

```
C = clone(H)
```

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ReceiverPreamp.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ReceiverPreamp` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.ReceiverPreamp.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose 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'`.

phased.ReceiverPreamp.step

Purpose

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

`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 `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

H

Receiver object.

X

Input signal.

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

Construct a receiver preamp object with a noise figure of 5 dB and bandwidth of 1 MHz. Demonstrate the effect of the receiver on a received sinusoid.

```
% construct receiver preamp object
hrx = phased.ReceiverPreamp('NoiseFigure',5,'SampleRate',1e6,...
    'NoiseBandwidth',1e6);
Fs = 1e3; t = linspace(0,1,1e3);
% signal at the receiver
x = cos(2*pi*200*t)';
% use the step method to obtain the signal demonstrating the
% effect of the receiver
y = step(hrx,x);
```

phased.RectangularWaveform

Purpose

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 3-658.
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.

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

Specify the sample rate, in hertz, as a positive scalar. The quantity `(SampleRate ./ PRF)` is a scalar or vector that must contain only integers. The default value of this property corresponds to 1 MHz.

Default: `1e6`

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`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency (in hertz) as a scalar or a row vector. The default value of this property corresponds to 10 kHz.

To implement a constant PRF, specify `PRF` as a positive scalar. To implement a staggered PRF, specify `PRF` as a row vector with positive elements. When `PRF` is a vector, the output pulses use successive elements of the vector as the PRF. If the last element of the vector is reached, the process continues cyclically with the first element of the vector.

The value of this property must satisfy these constraints:

- `PRF` is less than or equal to `(1/PulseWidth)`.
- `(SampleRate ./ PRF)` is a scalar or vector that contains only integers.

Default: `1e4`

OutputFormat

Output signal format

Specify the format of the output signal as one of `'Pulses'` or `'Samples'`. When you set the `OutputFormat` property to `'Pulses'`, the output of the `step` method is in the form of multiple pulses. In this case, the number of pulses is the value of the `NumPulses` property.

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: `'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`

Methods

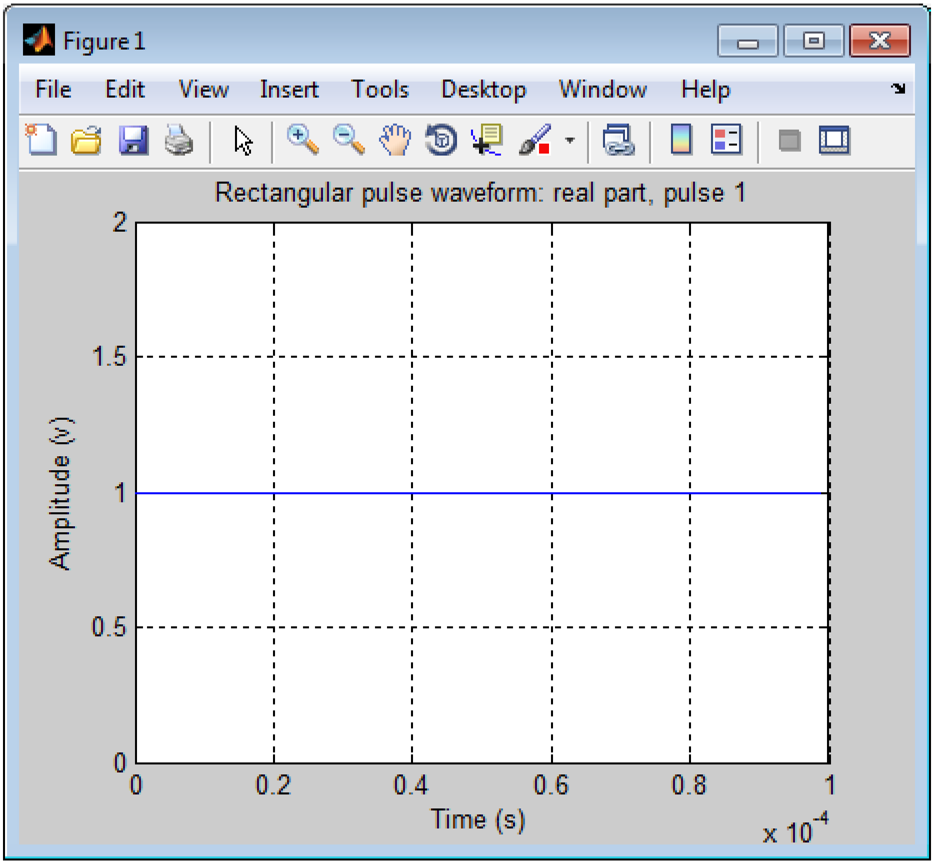
bandwidth	Bandwidth of rectangular pulse waveform
clone	Create rectangular waveform object with same property values
getMatchedFilter	Matched filter coefficients for waveform
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plot	Plot rectangular pulse waveform

release	Allow property value and input characteristics changes
reset	Reset states of rectangular waveform object
step	Samples of rectangular pulse waveform

Examples

Create and plot a rectangular pulse waveform object.

```
hw = phased.RectangularWaveform('PulseWidth',1e-4);
plot(hw);
```



References

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

See Also

`phased.LinearFMWaveform` | `phased.SteppedFMWaveform` | `phased.PhaseCodedWaveform` |

Related Examples

- Waveform Analysis Using the Ambiguity Function

phased.RectangularWaveform.bandwidth

Purpose

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

Determine the bandwidth of a rectangular pulse waveform.

```
H = phased.RectangularWaveform;
bw = bandwidth(H)
```

phased.RectangularWaveform.clone

Purpose Create rectangular waveform object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Matched filter coefficients for waveform

Syntax `Coeff = getMatchedFilter(H)`

Description `Coeff = getMatchedFilter(H)` returns the matched filter coefficients for the rectangular waveform object `H`. `Coeff` is a column vector.

Examples Get the matched filter coefficients for a rectangular pulse.

```
hw = phased.RectangularWaveform('PulseWidth',1e-5,...
    'OutputFormat','Pulses','NumPulses',1);
Coeff = getMatchedFilter(hw);
```

phased.RectangularWaveform.getNumInputs

Purpose Number of expected inputs to step method

Syntax

```
N = getNumInputs(H)
```

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.RectangularWaveform.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.RectangularWaveform.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `RectangularWaveform` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose

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

String that specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `Type` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Default: `'b'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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`

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

Create and plot a rectangular pulse waveform.

```
hw = phased.RectangularWaveform('PulseWidth',1e-4);
plot(hw);
```

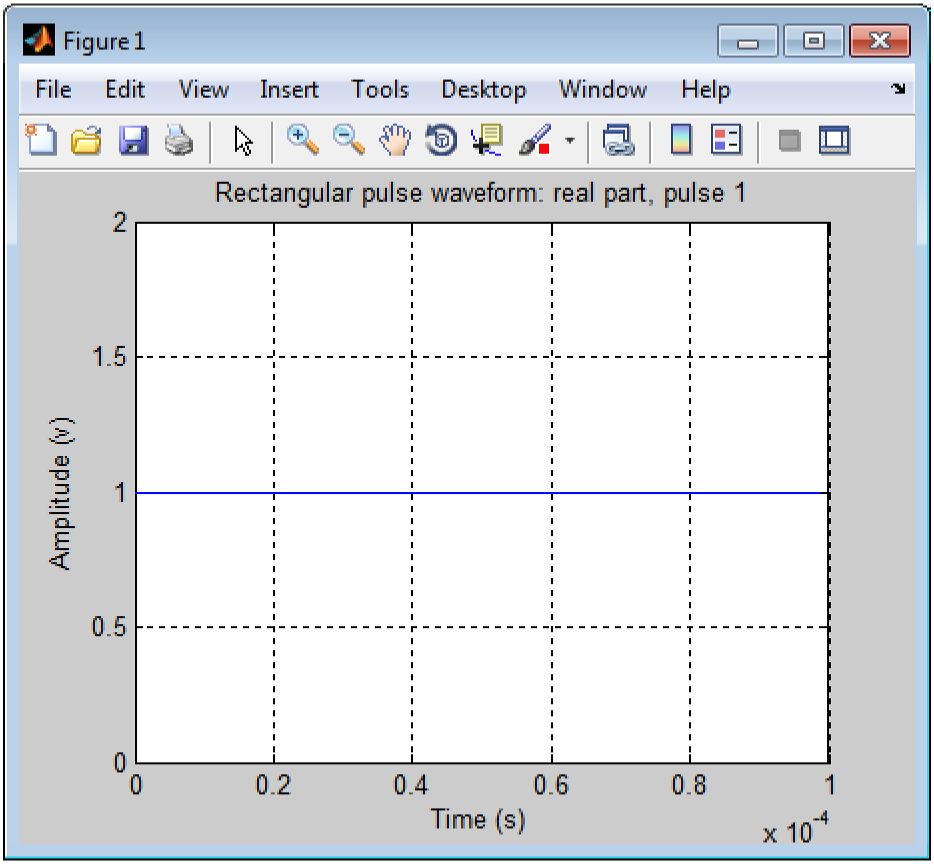


Figure 1
File Edit View Insert Tools Desktop Window Help
Rectangular pulse waveform: real part, pulse 1
Amplitude (v)
0
0.5
1
1.5
2
Time (s)
0
0.2
0.4
0.6
0.8
1
x 10^-4

phased.RectangularWaveform.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

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.

phased.RectangularWaveform.step

Purpose Samples of rectangular pulse waveform

Syntax `Y = step(H)`

Description `Y = step(H)` returns samples of the rectangular pulse in a column vector `Y`.

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.

Examples Construct a rectangular pulse 10 microseconds in duration with pulse repetition interval of 100 microseconds.

```
hw = phased.RectangularWaveform('PulseWidth',1e-5,...
           'OutputFormat','Pulses','NumPulses',1,...
           'SampleRate',1e6,'PRF',1e4);
wav = step(hw);
```

Purpose

Phased array formed by replicated subarrays

Description

The `ReplicatedSubarray` object represents a phased array that contains copies of a subarray.

To obtain the response of the subarrays:

1. Define and set up your phased array containing replicated subarrays. See "Construction" on page 3-675.
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.

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 scalar or length-2 row vector. This property applies 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 length-2 row vector, the first entry is the number of subarrays in each row. The second entry is the number of subarrays in each column. The row is along the local *y*-axis, and the column is along the local *z*-axis.

Default: `[2 1]`

GridSpacing

Spacing of rectangular grid

Specify the rectangular grid spacing of the array in meters, as a scalar, length-2 row vector, or the string value `'Auto'`. This property applies when you set the `Layout` property to `'Rectangular'`.

If `GridSpacing` is a scalar, the spacing along the row and the spacing along the column are the same.

If `GridSpacing` is a length-2 row vector, the first entry specifies the spacing along the row. The second entry specifies the spacing along the column.

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 steering the subarray as one of `'None' | 'Phase' | 'Time'`.

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`

Methods

clone	Create replicated subarray with same property values
collectPlaneWave	Simulate received plane waves
getElementPosition	Positions of array elements
getNumElements	Number of elements in array
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
getNumSubarrays	Number of subarrays in array

getSubarrayPosition	Positions of subarrays in array
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of array
release	Allow property value and input characteristics changes
step	Output responses of subarrays
viewArray	View array geometry

Examples

Azimuth Response of Array with Subarrays

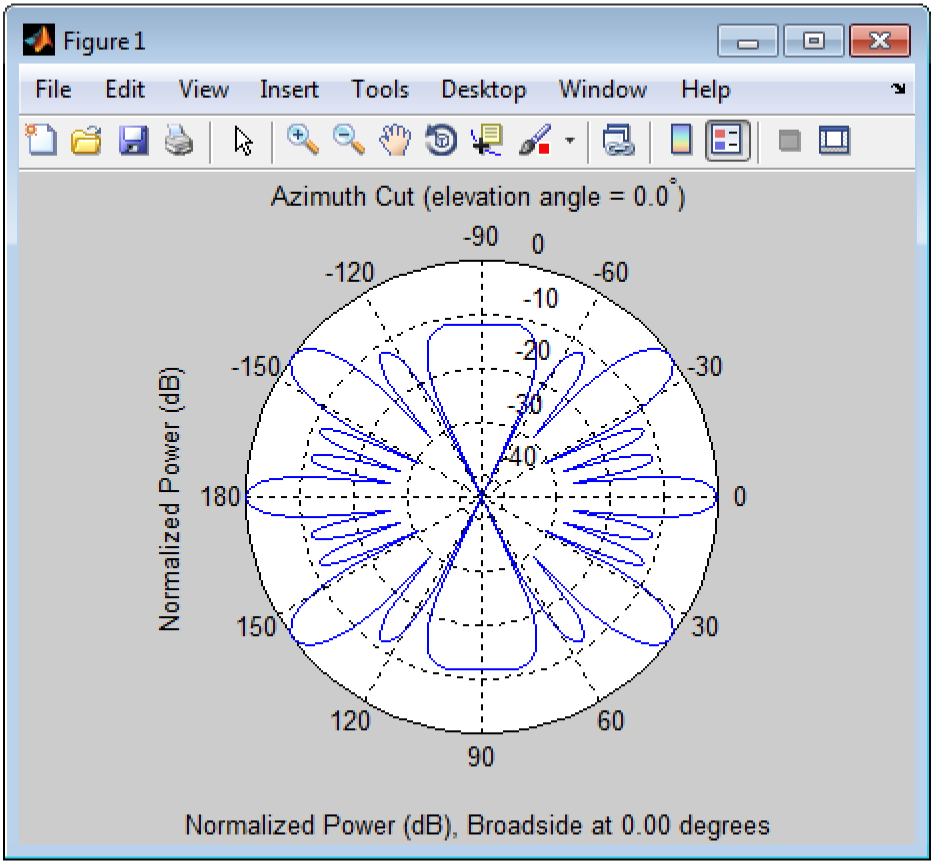
Plot the azimuth response of a 4-element ULA composed of two 2-element ULAs.

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',[2 1],...
   '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');
```



Response of Subarrays

Calculate the response at the boresight of two 2-element ULAs that are subarrays of a 4-element ULA.

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',[2 1],...
   'GridSpacing','Auto');
```

Find the response of each subarray at the boresight. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

```
RESP = step(ha,1e9,[0;0],3e8);
```

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.

See Also

`phased.ULA` | `phased.URA` | `phased.ConformalArray` | `phased.PartitionedArray` |

Related Examples

- Subarrays in Phased Array Antennas
- Phased Array Gallery

Concepts

- “Subarrays Within Arrays”

phased.ReplicatedSubarray.clone

Purpose Create replicated subarray with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose

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)` uses `FREQ` as the incoming signal's carrier frequency.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)` uses `C` as the signal's propagation speed. `C` must be a scalar.

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 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, 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 composed of four 4-element ULAs.

Create a 4-element ULA, and replicate it to create a 16-element ULA.

```
hs = phased.ULA('NumElements',4);
ha = phased.ReplicatedSubarray('Subarray',hs,...
   'GridSize',[4 1]);
```

Simulate receiving signals from 10 degrees 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.

```
Y = collectPlaneWave(ha,randn(4,2),[10 30],...
   1e8,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`

phased.ReplicatedSubarray.getElementPosition

Purpose

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.

```
H = phased.ReplicatedSubarray('Subarray',...
    phased.ULA('NumElements',3),'GridSize',[1 2]);
POS = getElementPosition(H)
```

See Also

`getSubarrayPosition` |

phased.ReplicatedSubarray.getNumElements

Purpose

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 ReplicatedSubarrays

Create an array with two copies of a 3-element ULA, and obtain the total number of elements.

```
H = phased.ReplicatedSubarray('Subarray',...
    phased.ULA('NumElements',3),'GridSize',[1 2]);
N = getNumElements(H);
```

See Also

`getNumSubarrays` |

phased.ReplicatedSubarray.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.ReplicatedSubarray.getNumSubarrays

Purpose 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. Obtain the number of subarrays.

```
H = phased.ReplicatedSubarray('Subarray',...
    phased.ULA('NumElements',3),'GridSize',[2 5]);
N = getNumSubarrays(H);
```

See Also `getNumElements` **|**

phased.ReplicatedSubarray.getSubarrayPosition

Purpose

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

Positions of Replicated Subarrays in Array

Create an array with two copies of a 3-element ULA, and obtain the positions of the subarrays.

```
H = phased.ReplicatedSubarray('Subarray',...
    phased.ULA('NumElements',3),'GridSize',[1 2]);
POS = getSubarrayPosition(H)
```

See Also

`getElementPosition` |

phased.ReplicatedSubarray.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ReplicatedSubarray` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose

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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can

specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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'`, and `'pow'`.

Default: `'db'`

Weights

Weights applied to the array, specified as a length-N column vector or N-by-M matrix. N is the number of subarrays in the array. M is the number of frequencies in `FREQ`. If `Weights` is a vector, the function applies the same weights to each frequency. If

`Weights` is a matrix, the function applies each column of weight values to the corresponding frequency in `FREQ`.

Examples

Azimuth Response of Array with Subarrays

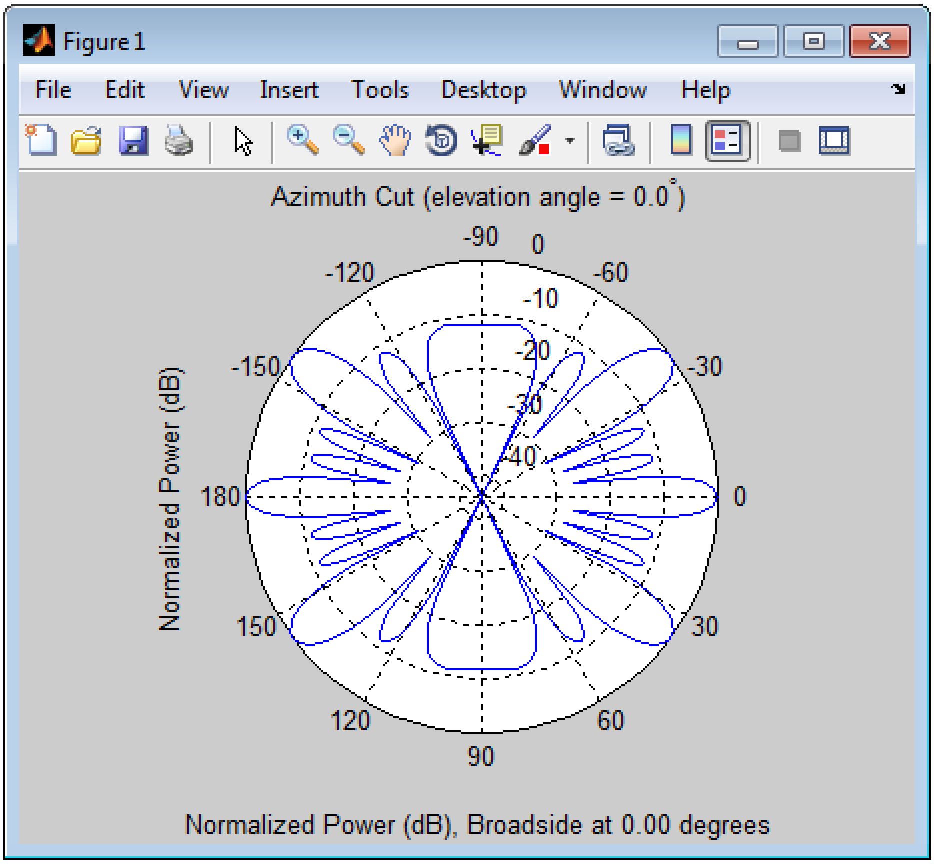
Plot the azimuth response of a 4-element ULA composed of two 2-element ULAs.

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',[2 1],...
   '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');
```



See Also

`uv2azel` | `azel2uv`

phased.ReplicatedSubarray.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Output responses of subarrays

Syntax

```
RESP = step(H,FREQ,ANG,V)
RESP = step(H,FREQ,ANG,V,STEERANGLE)
```

Description

`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 subarray's steering direction. This syntax is available when you set the SubarraySteering property to either `'Phase'` or `'Time'`.

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

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 `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.

Output Arguments

RESP

Responses of subarrays of array. `RESP` has dimensions N-by-M-by-L. N is the number of subarrays in the phased array. 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`.

Examples

Response of Subarrays

Calculate the response at the boresight of two 2-element ULAs that are subarrays of a 4-element ULA.

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',[2 1],...
   'GridSpacing','Auto');
```

Find the response of each subarray at the boresight. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

```
RESP = step(ha,1e9,[0;0],3e8);
```

See Also

`uv2azel` | `phitheta2azel`

phased.ReplicatedSubarray.viewArray

Purpose

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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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 string `'All'` to show indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

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`

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 string `'All'` to highlight all subarrays of the array or `'None'` to suppress the subarray highlighting. The highlighting uses different colors for different subarrays.

Default: `'All'`

Title

String 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

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);
```

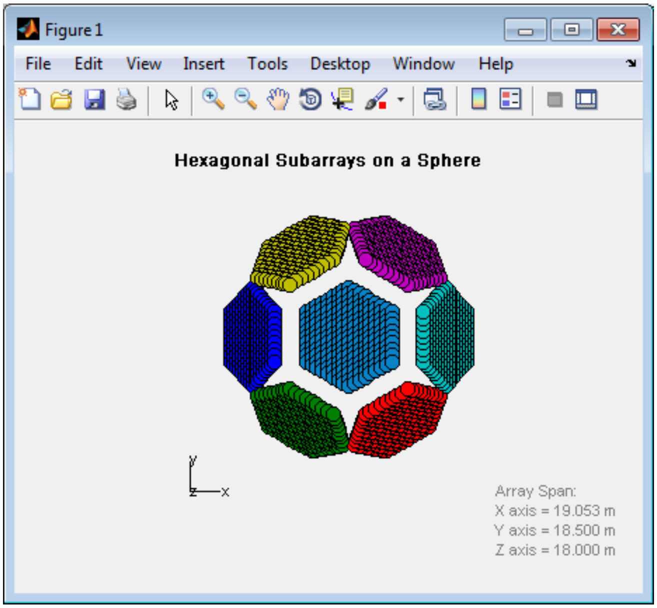
```
rowidx = 0;
for m = Nmin-Nmax:Nmax-Nmin
   rowidx = rowidx+1;
   idx = startvals(rowidx):stopvals(rowidx);
   pos(2,idx) = (-(rowlengths(rowidx)-1)/2:...
                 (rowlengths(rowidx)-1)/2) * dy;
   pos(3,idx) = m * dz;
end
hexa = phased.ConformalArray('ElementPosition',pos,...
   'ElementNormal',zeros(2,numels_hex));
```

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(degtorad(az),degtorad(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` |

Related Examples

- Phased Array Gallery

phased.RootMUSICEstimator

Purpose

Root MUSIC direction of arrival (DOA) estimator

Description

The `RootMUSICEstimator` object implements a root multiple signal classification (MUSIC) direction of arrival estimate for a uniform linear array.

To estimate the direction of arrival (DOA):

1. Define and set up your DOA estimator. See "Construction" on page 3-706.
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.

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

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`

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. The default value indicates no spatial smoothing.

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.

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'`.

Default: `1`

Methods

clone	Create root MUSIC DOA estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties

release	Allow property value and input characteristics changes
step	Perform DOA estimation

Examples

Estimate the DOAs 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 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);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
hdoa = phased.RootMUSICEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property','NumSignals',2);
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60])
```

References

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

See Also

`broadside2azphased.RootWSFEstimator` |

phased.RootMUSICEstimator.clone

Purpose Create root MUSIC DOA estimator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.RootMUSICEstimator.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.RootMUSICEstimator.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.RootMUSICEstimator.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `RootMUSICEstimator` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.RootMUSICEstimator.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform DOA estimation

Syntax

```
ANG = step(H,X)
```

Description

`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).

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.

Examples

Estimate the DOAs 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 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);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
hdoa = phased.RootMUSICEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property','NumSignals',2);
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60])
```

phased.RootWSFEstimator

Purpose

Root WSF direction of arrival (DOA) estimator

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 3-716.
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.

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.

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`

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'`.

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.

Default: `'Inf'`

Methods

clone	Create root WSF DOA estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform DOA estimation

Examples

Estimate the DOAs 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 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);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
hdoa = phased.RootWSFEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property','NumSignals',2);
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60])
```

References

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

See Also

`broadside2azphased.RootMUSICEstimator` |

phased.RootWSFEstimator.clone

Purpose Create root WSF DOA estimator object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.RootWSFEstimator.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.RootWSFEstimator.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `RootWSFEstimator` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.RootWSFEstimator.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.RootWSFEstimator.step

Purpose

Perform DOA estimation

Syntax

`ANG = step(H,X)`

Description

`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).

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.

Examples

Estimate the DOAs 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 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);
ha = phased.ULA('NumElements',10,'ElementSpacing',1);
ha.Element.FrequencyRange = [100e6 300e6];
fc = 150e6;
x = collectPlaneWave(ha,[x1 x2],[10 20;45 60]',fc);
rng default;
noise = 0.1/sqrt(2)*(randn(size(x))+1i*randn(size(x)));
hdoa = phased.RootWSFEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property','NumSignals',2);
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60])
```

phased.STAPSMIBeamformer

Purpose

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 3-726.

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.

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

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 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

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`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency (PRF) of the received signal in hertz as a scalar.

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'`.

Default: `[0; 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'`.

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`

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.

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.

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

Methods

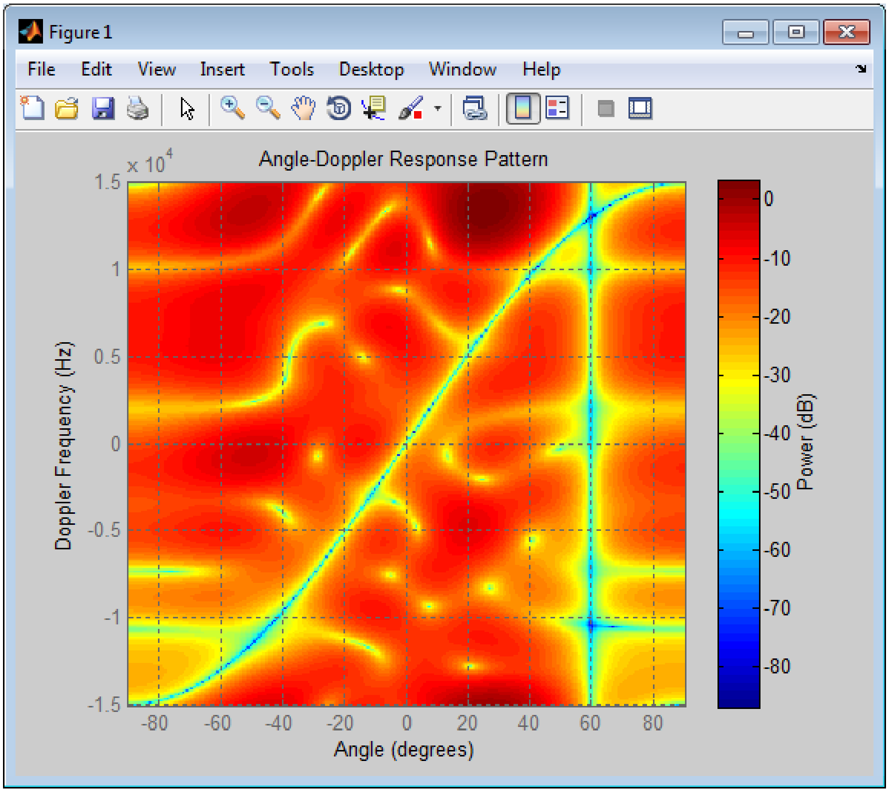
clone	Create space-time adaptive SMI beamformer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method

isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform SMI STAP processing on input data

Examples

Process the data cube using an SMI processor. The weights are calculated for the 71st cell of a collected data cube pointing to the direction of [45; –35] degrees and the Doppler of 12980 Hz.

```
load STAPExampleData;   % load data
Hs = 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] = step(Hs,STAPEx_ReceivePulse,71,[45; -35],12980);
Hresp = phased.AngleDopplerResponse(...
    'SensorArray',Hs.SensorArray,...
    'OperatingFrequency',Hs.OperatingFrequency,...
    'PRF',Hs.PRF,...
    'PropagationSpeed',Hs.PropagationSpeed);
plotResponse(Hresp,w);
```



Algorithms

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.

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.

See Also

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

phased.STAPSMIBeamformer.clone

Purpose

Create space-time adaptive SMI beamformer object with same property values

Syntax

`C = clone(H)`

Description

`C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.STAPSMIBeamformer.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.STAPSMIBeamformer.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.STAPSMIBeamformer.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `STAPSMIBeamformer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.STAPSMIBeamformer.release

Purpose

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform SMI STAP processing on input data

Syntax

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

Description

`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'`.

`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 `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.

Examples

Process the data cube using an SMI processor. The weights are calculated for the 71st cell of a collected data cube pointing to the direction of [45; –35] degrees and the Doppler of 12980 Hz.

```
load STAPExampleData;   % load data
Hs = 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] = step(Hs,STAPEx_ReceivePulse,71,[45; -35],12980);
```

See Also

`uv2azel | phitheta2azel`

Purpose

Sensor array steering vector

Description

The `SteeringVector` object calculates the steering vector for a sensor array.

To compute the steering vector of the array for specified directions:

1. Define and set up your steering vector calculator. See "Construction" on page 3-741.
2. Call `step` to compute the steering vector according to the properties of `phased.SteeringVector`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.SteeringVector` creates a steering vector System object, `H`. The object calculates the steering vector of the given sensor array for the specified directions.

`H = phased.SteeringVector(Name,Value)` creates a steering vector 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 steering vector

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

IncludeElementResponse

Include individual element response in the steering vector

If this property is `true`, the steering vector includes the individual element responses.

If this property is `false`, the computation of the steering vector assumes the elements are isotropic. The steering vector does not include the individual element responses. Furthermore, if the `SensorArray` property contains subarrays, the steering vector is the array factor among the subarrays. If `SensorArray` does not contain subarrays, the steering vector is the array factor among the array elements.

Default: `false`

Methods

clone	Create steering vector object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Calculate steering vector

Examples

Steering Vector for Uniform Linear Array

Calculate the steering vector for a uniform linear array at the direction of 30 degrees azimuth and 20 degrees elevation. Assume the array's operating frequency is 300 MHz.

```
hULA = phased.ULA('NumElements',2);
hsv = phased.SteeringVector('SensorArray',hULA);
Fc = 3e8;
ANG = [30; 20];
sv = step(hsv,Fc,ANG);
```

Beam Pattern Before and After Steering

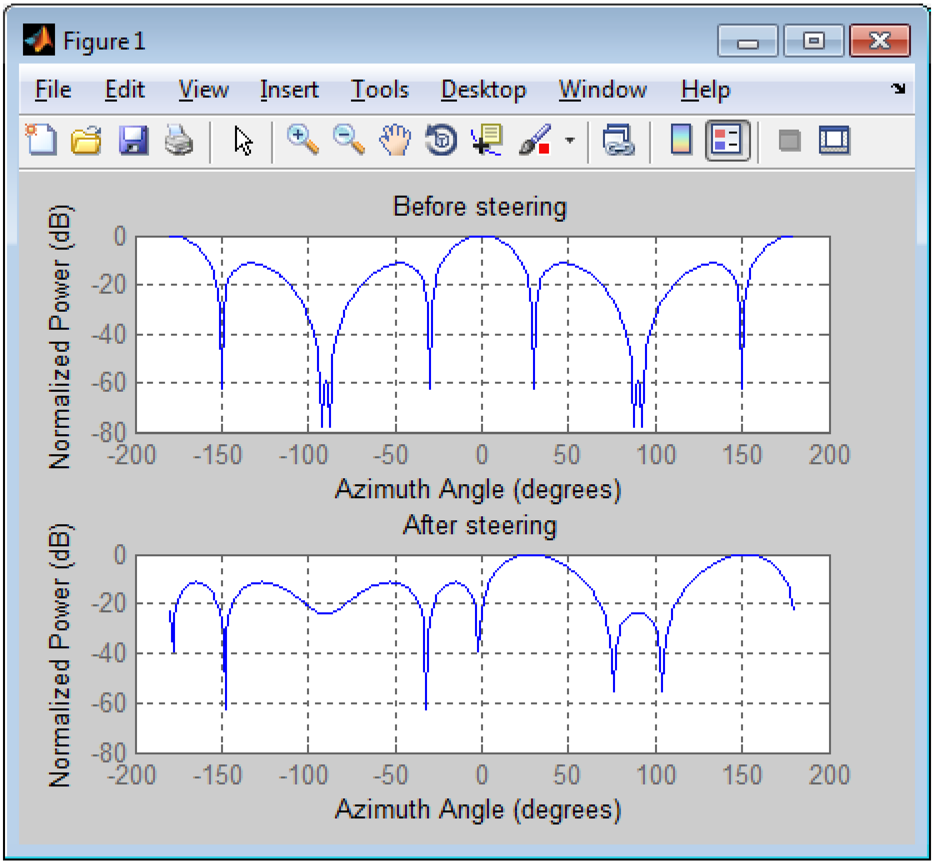
Plot the beam pattern for a uniform linear array before and after steering.

Calculate the steering vector for a 4-element uniform linear array at the direction of 30 degrees azimuth and 20 degrees elevation. Assume the array's operating frequency is 300 MHz.

```
ha = phased.ULA('NumElements',4);
hsv = phased.SteeringVector('SensorArray',ha);
sv = step(hsv,3e8,[30; 20]);
```

Compare the beam pattern before and after the steering.

```
c = hsv.PropagationSpeed;
subplot(211)
plotResponse(ha,3e8,c,'RespCut','Az');
title('Before steering');
subplot(212)
plotResponse(ha,3e8,c,'RespCut','Az','Weights',sv);
title('After steering');
```



References

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

See Also

`phased.ArrayGain` | `phased.ArrayResponse` | `phased.ElementDelay` |

Purpose Create steering vector object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.SteeringVector.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.SteeringVector.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.SteeringVector.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `SteeringVector` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.SteeringVector.step

Purpose

Calculate steering vector

Syntax

```
SV = step(H,FREQ,ANG)
SV = step(H,FREQ,ANG,STEERANGLE)
```

Description

`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`.

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

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 and 180 degrees, and the elevation angle must be between –90 and 90 degrees.

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.

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.

Output Arguments

SV

Steering vector. `SV` has dimensions N-by-M-by-L. N is the number of subarrays in the phased array if `H.SensorArray` contains subarrays, or the number of elements otherwise. 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`.

Examples

Steering Vector for Uniform Linear Array

Calculate the steering vector for a uniform linear array at the direction of 30 degrees azimuth and 20 degrees elevation. Assume the array's operating frequency is 300 MHz.

```
hULA = phased.ULA('NumElements',2);
hsv = phased.SteeringVector('SensorArray',hULA);
Fc = 3e8;
ANG = [30; 20];
sv = step(hsv,Fc,ANG);
```

See Also

`uv2azel` | `phitheta2azel`

Purpose

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 3-753.
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.

Construction

`H = phased.SteppedFMWaveform` creates a stepped FM pulse waveform System object, `H`. The object generates samples of a linearly stepped FM pulse waveform.

`H = phased.SteppedFMWaveform(Name,Value)` creates a stepped FM 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

Specify the sample rate, in hertz, as a positive scalar. The quantity `(SampleRate ./ PRF)` is a scalar or vector that must contain only integers. The default value of this property corresponds to 1 MHz.

Default: `1e6`

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`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency (in hertz) as a scalar or a row vector. The default value of this property corresponds to 10 kHz.

To implement a constant PRF, specify `PRF` as a positive scalar. To implement a staggered PRF, specify `PRF` as a row vector with positive elements. When `PRF` is a vector, the output pulses use successive elements of the vector as the PRF. If the last element of the vector is reached, the process continues cyclically with the first element of the vector.

The value of this property must satisfy these constraints:

- `PRF` is less than or equal to `(1/PulseWidth)`.
- `(SampleRate ./ PRF)` is a scalar or vector that contains only integers.

Default: `1e4`

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: `2e4`

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`

OutputFormat

Output signal format

Specify the format of the output signal as one of `'Pulses'` or `'Samples'`. When you set the `OutputFormat` property to `'Pulses'`, the output of the `step` method is in the form of multiple pulses. In this case, the number of pulses is the value of the `NumPulses` property.

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: `'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`

Methods

bandwidth	Bandwidth of stepped FM pulse waveform
clone	Create stepped FM pulse waveform object with same property values
getMatchedFilter	Matched filter coefficients for waveform
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plot	Plot stepped FM pulse waveform
release	Allow property value and input characteristics changes
reset	Reset state of stepped FM pulse waveform object
step	Samples of stepped FM pulse waveform

Definitions

Stepped FM Waveform

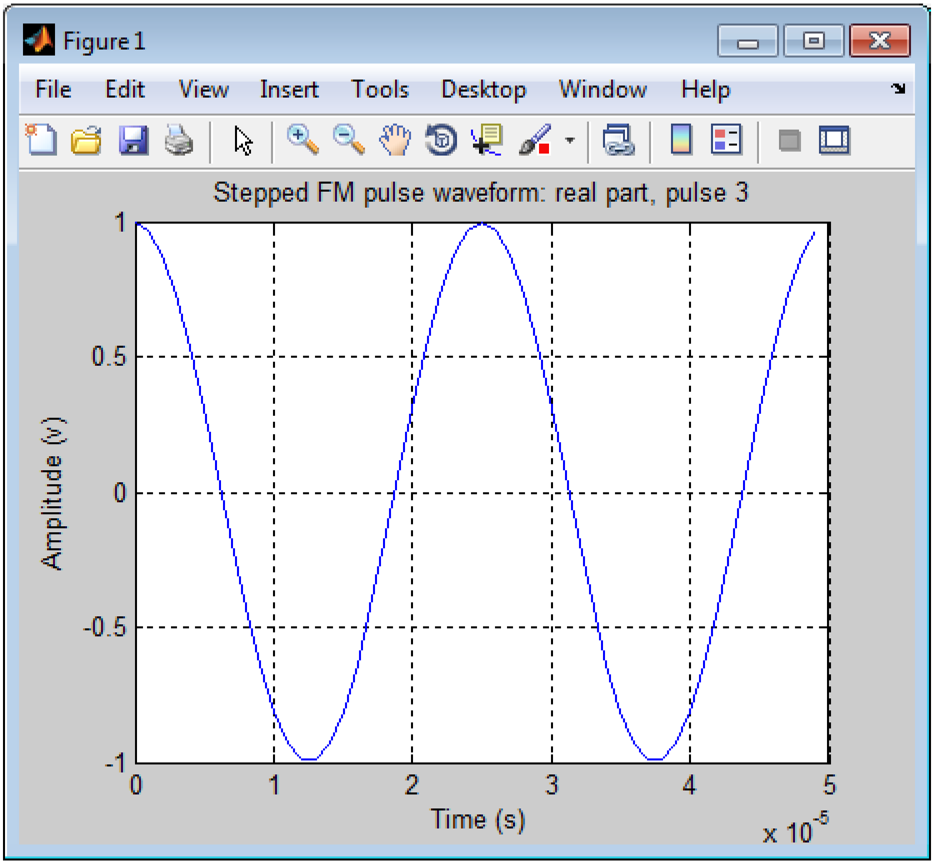
In a stepped FM waveform, a group of pulses together sweep a certain bandwidth. Each pulse in this group occupies a given center frequency and these center frequencies are uniformly located within the total bandwidth.

Examples

Create a stepped frequency pulse waveform object, and plot the third pulse.

```
hw = phased.SteppedFMWaveform('NumSteps',3,'FrequencyStep',2e4);
```

```
plot(hw,'PulseIdx',3);
```



References

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

See Also

`phased.LinearFMWaveform` | `phased.RectangularWaveform` | `phased.PhaseCodedWaveform` |

Related Examples

- Waveform Analysis Using the Ambiguity Function

phased.SteppedFMWaveform.bandwidth

Purpose

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

Determine the bandwidth of a stepped FM waveform.

```
H = phased.SteppedFMWaveform;
bw = bandwidth(H)
```

phased.SteppedFMWaveform.clone

Purpose Create stepped FM pulse waveform object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Matched filter coefficients for waveform

Syntax

```
Coeff = getMatchedFilter(H)
```

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.

Examples Get the matched filter coefficients for a stepped FM pulse waveform.

```
hw = phased.SteppedFMWaveform(...
    'NumSteps',3,'FrequencyStep',2e4,...
    'OutputFormat','Pulses','NumPulses',3);
coeff = getMatchedFilter(hw);
```

phased.SteppedFMWaveform.getNumInputs

Purpose Number of expected inputs to step method

Syntax

```
N = getNumInputs(H)
```

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.SteppedFMWaveform.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.SteppedFMWaveform.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `SteppedFMWaveform` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose

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

String that specifies the same line color, style, or marker options as are available in the MATLAB `plot` function. If you specify a `Type` value of `'complex'`, then `LineSpec` applies to both the real and imaginary subplots.

Default: `'b'`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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`

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

Create and plot a stepped frequency pulse waveform.

```
hw = phased.SteppedFMWaveform;
plot(hw);
```

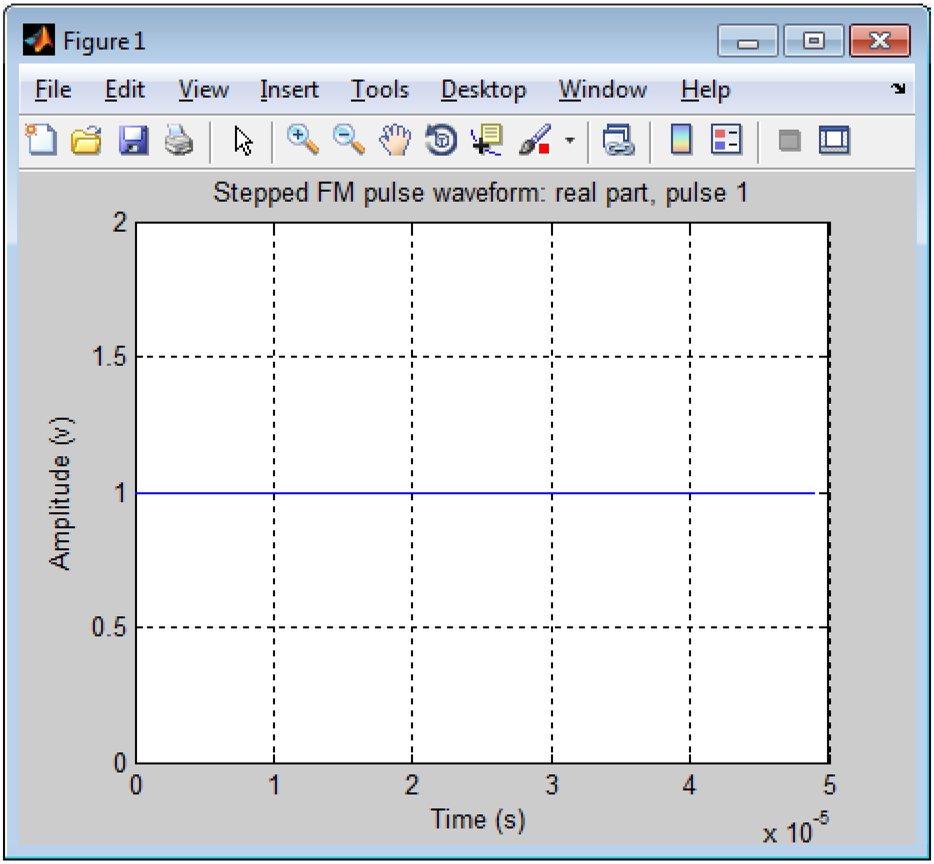


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Stepped FM pulse waveform: real part, pulse 1
Amplitude (v)
2
1.5
1
0.5
0
0
1
2
3
4
5
Time (s)
x 10^-5

phased.SteppedFMWaveform.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose 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.

phased.SteppedFMWaveform.step

Purpose

Samples of stepped FM pulse waveform

Syntax

`Y = step(H)`

Description

`Y = step(H)` 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.

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.

Definitions

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.

Examples

Create a stepped frequency pulse waveform object with a frequency step of 20 kHz and three frequency steps.

```
hw = phased.SteppedFMWaveform(...
    'NumSteps',3,'FrequencyStep',2e4,...
    'OutputFormat','Pulses','NumPulses',1);
% Use the step method to obtain the pulses.
% Pulse 1
pulse1 = step(hw);
```

```
% Pulse 2 incremented by the frequency step 20 kHz
pulse2 = step(hw);
% Pulse 3 incremented by the frequency step 20 kHz
pulse3 = step(hw);
```

phased.StretchProcessor

Purpose

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 3-772.
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.

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, 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 quantity `(SampleRate ./ PRF)` is a scalar or vector that must contain only integers. The default value of this property corresponds to 1 MHz.

Default: `1e6`

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`

PRF

Pulse repetition frequency

Specify the pulse repetition frequency (in hertz) as a scalar or a row vector. The default value of this property corresponds to 10 kHz.

To implement a constant PRF, specify `PRF` as a positive scalar. To implement a staggered PRF, specify `PRF` as a row vector with positive elements. When `PRF` is a vector, the output pulses use successive elements of the vector as the PRF. If the last element of the vector is reached, the process continues cyclically with the first element of the vector.

The value of this property must satisfy these constraints:

- `PRF` is less than or equal to `(1/PulseWidth)`.
- `(SampleRate ./ PRF)` is a scalar or vector that contains only integers.

Default: `1e4`

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 sweeping bandwidth. If `SweepInterval` is `'Symmetric'`, the waveform sweeps in the interval between –B/2 and B/2.

Default: `'Positive'`

PropagationSpeed

Signal propagation speed

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

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 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.

Default: `500`

Methods

clone	Create stretch processor with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform stretch processing for linear FM waveform

Examples

Detection of Target Using Stretch Processing

Use stretch processing to locate a target at a range of 4950 m.

Simulate the signal.

```
hwav = phased.LinearFMWaveform;
x = step(hwav);
c = 3e8; r = 4950;
num_sample = r/(c/(2*hwav.SampleRate));
x = circshift(x,num_sample);
```

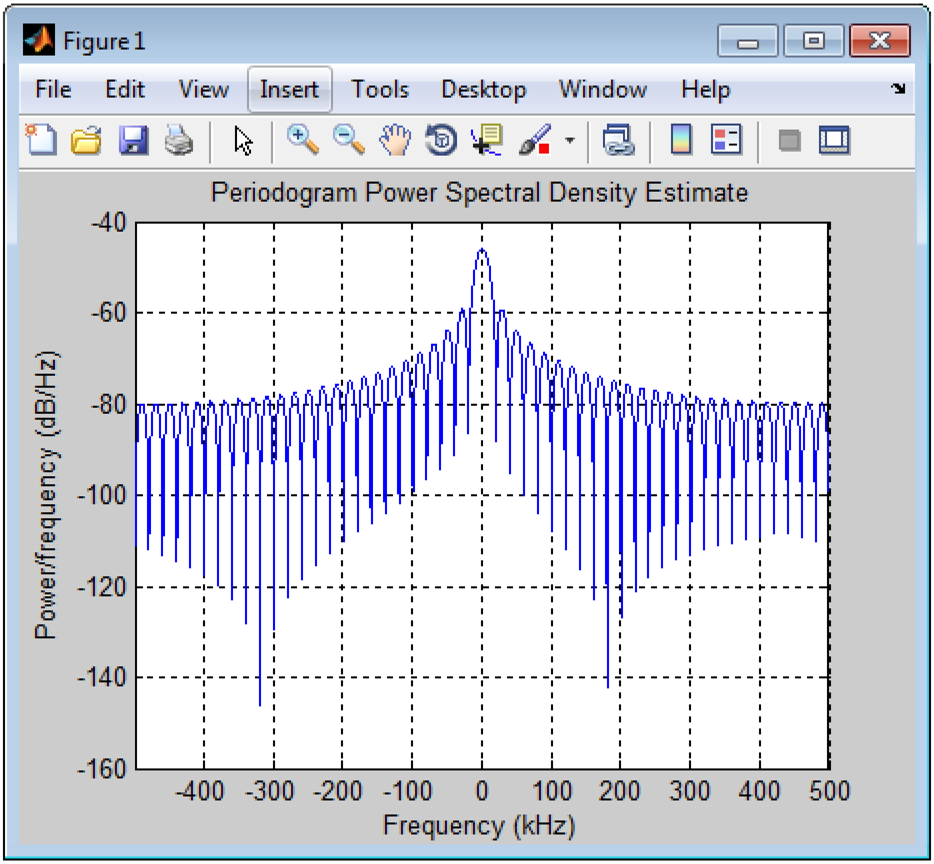
Perform stretch processing.

```
hs = getStretchProcessor(hwav,5000,200,c);
y = step(hs,x);
```

Plot the spectrum of the resulting signal.

```
hp = spectrum.periodogram;
hpsd = psd(hp,y,'Fs',hs.SampleRate,'NFFT',2048,...
   'CenterDC',true);
```

```
plot(hpsd);
```



Detect the range.

```
[~,rngidx] = findpeaks(pow2db(hpsd.Data/max(hpsd.Data)),...
   'MinPeakHeight',-5);
rngfreq = hpsd.Frequencies(rngidx);
re = stretchfreq2rng(rngfreq,hs.SweepSlope,...
   hs.ReferenceRange,c);
```

References

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

See Also

`phased.LinearFMWaveform` | `phased.MatchedFilter` | `stretchfreq2rng`

Related Examples

- Range Estimation Using Stretch Processing

Concepts

- "Stretch Processing"

phased.StretchProcessor.clone

Purpose Create stretch processor with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.StretchProcessor.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.StretchProcessor.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.StretchProcessor.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `StretchProcessor` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.StretchProcessor.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform stretch processing for linear FM waveform

Syntax

```
Y = step(H,X)
```

Description

`Y = step(H,X)` applies stretch processing along the first dimension of `X`. Each column of `X` represents one receiving pulse.

Input Arguments

H

Stretch processor object.

X

Input signal. Each column represents one receiving pulse.

Output Arguments

Y

Result of stretch processing. The dimensions of `Y` match the dimensions of `X`.

Examples

Detection of Target Using Stretch Processing

Use stretch processing to locate a target at a range of 4950 m.

Simulate the signal.

```
hwav = phased.LinearFMWaveform;
x = step(hwav);
c = 3e8; r = 4950;
num_sample = r/(c/(2*hwav.SampleRate));
x = circshift(x,num_sample);
```

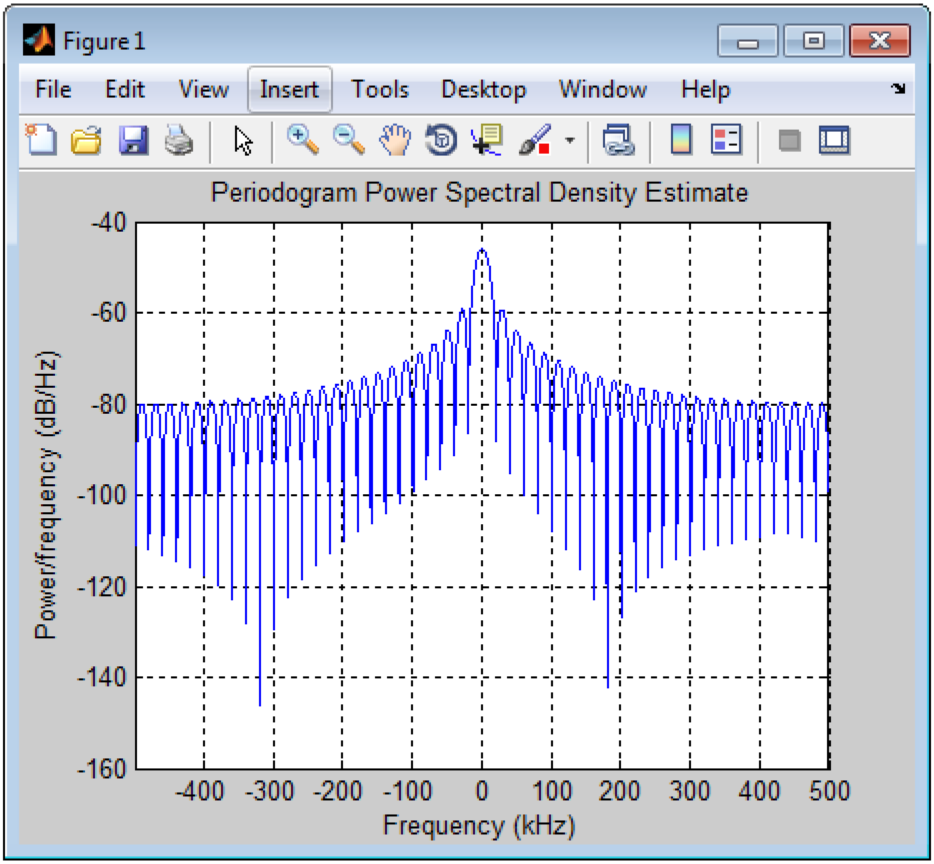
Perform stretch processing.

```
hs = getStretchProcessor(hwav,5000,200,c);
y = step(hs,x);
```

Plot the spectrum of the resulting signal.

```
hp = spectrum.periodogram;
```

```
hpsd = psd(hp,y,'Fs',hs.SampleRate,'NFFT',2048,...
    'CenterDC',true);
plot(hpsd);
```



Detect the range.

```
[~,rngidx] = findpeaks(pow2db(hpsd.Data/max(hpsd.Data)),...
    'MinPeakHeight',-5);
rngfreq = hpsd.Frequencies(rngidx);
re = stretchfreq2rng(rngfreq,hs.SweepSlope,...
```

```
              hs.ReferenceRange,c);
```

See Also

`stretchfreq2rng`

Related Examples

- Range Estimation Using Stretch Processing

Concepts

- “Stretch Processing”

phased.SubbandPhaseShiftBeamformer

Purpose

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 3-786.
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.

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

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 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

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.

Default: `3e8`

SampleRate

Signal sampling rate

Specify the signal sampling rate (in hertz) as a positive scalar.

Default: `1e6`

NumSubbands

Number of subbands

Specify the number of subbands used in the subband processing as a positive integer.

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'`.

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

clone	Create subband phase shift beamformer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Beamforming using subband phase shifting

Examples

Apply subband phase shift beamformer to an 11-element ULA. The incident angle of the signal is 10 degrees in azimuth and 30 degrees in elevation.

```
% Signal simulation
ha = phased.ULA('NumElements',11,'ElementSpacing',0.3);
ha.Element.FrequencyRange = [20 20000];
fs = 1e3; carrierFreq = 2e3; t = (0:1/fs:2)';
x = chirp(t,0,2,fs);
c = 1500; % Wave propagation speed (m/s)
hc = phased.WidebandCollector('Sensor',ha,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',true,'CarrierFrequency',carrierFreq);
incidentAngle = [10; 30];
```

```
x = step(hc,x,incidentAngle);
noise = 0.3*(randn(size(x)) + 1j*randn(size(x)));
rx = x+noise;

% Beamforming
hbf = phased.SubbandPhaseShiftBeamformer('SensorArray',ha,...
    'Direction',incidentAngle,...
    'OperatingFrequency',carrierFreq,'PropagationSpeed',c,...
    'SampleRate',fs,'SubbandsOutputPort',true,...
    'WeightsOutputPort',true);
[y,w,subbandfreq] = step(hbf,rx);

% Plot 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 response pattern for five bands
figure;
plotResponse(ha,subbandfreq(1:5).',c,'Weights',w(:,1:5));
legend('location','SouthEast')
```

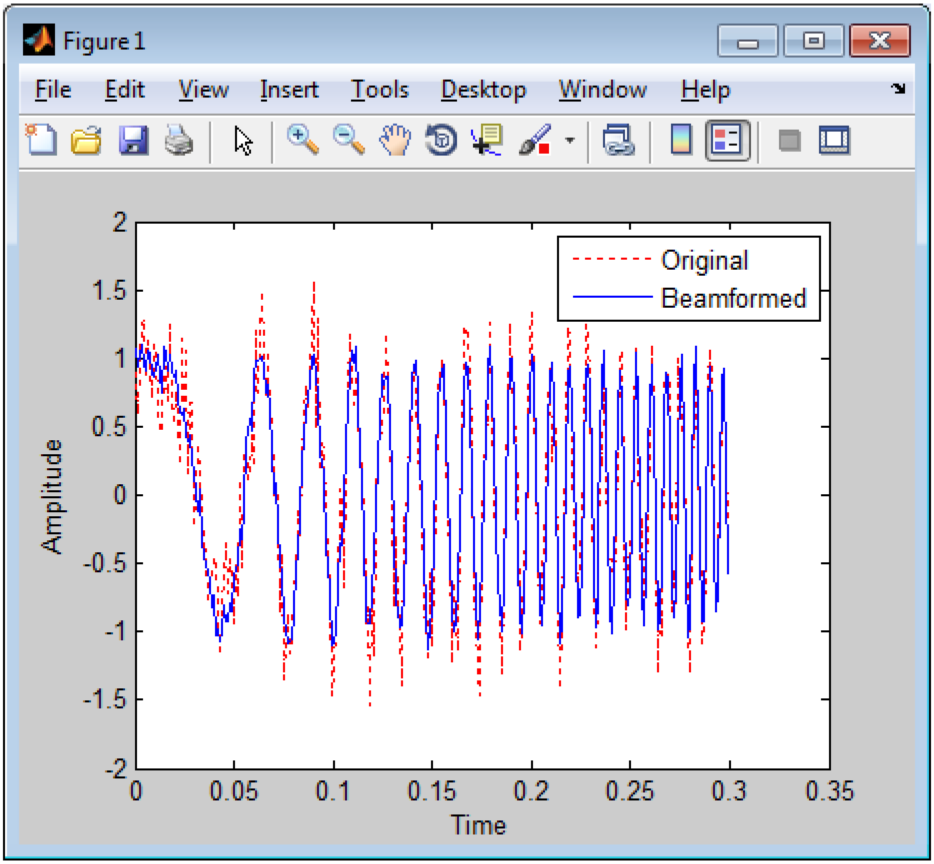
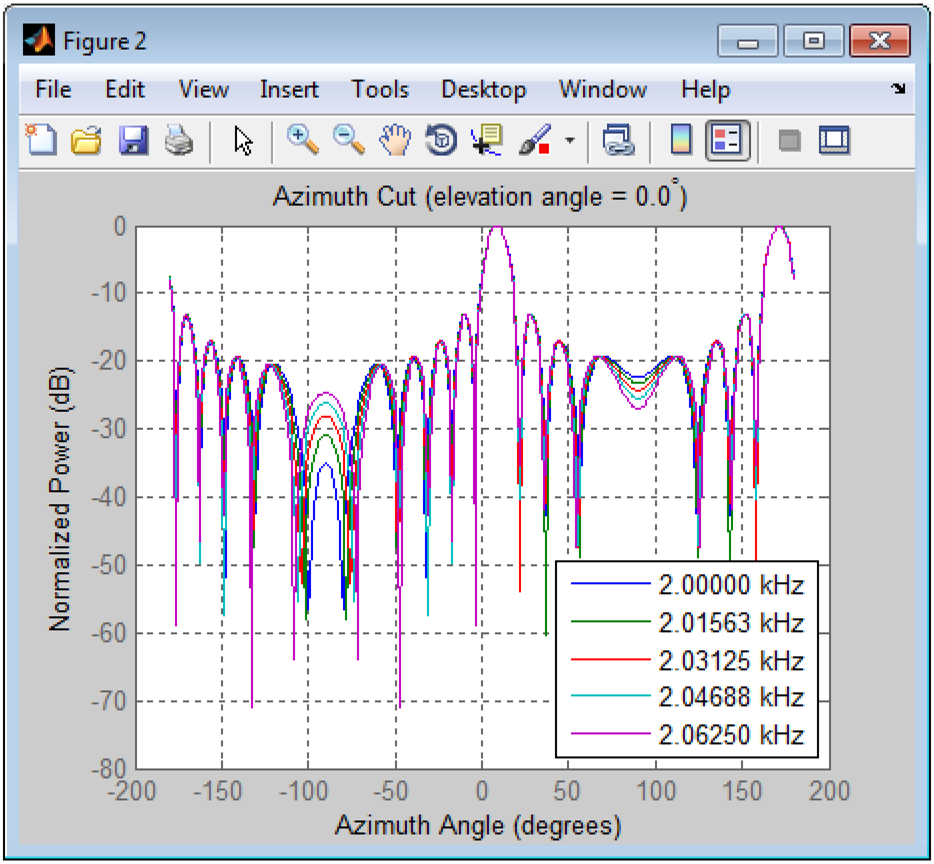


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Original
Beamformed
Amplitude
Time
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35



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

`phased.Collector` | `phased.PhaseShiftBeamformer` | `phased.TimeDelayBeamformer` | `phased.WidebandCollector` | `uv2azel` | `phitheta2azel`

Related Examples

- "Wideband Beamforming"

phased.SubbandPhaseShiftBeamformer.clone

Purpose Create subband phase shift beamformer object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.SubbandPhaseShiftBeamformer.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.SubbandPhaseShiftBeamformer.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `SubbandPhaseShiftBeamformer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.SubbandPhaseShiftBeamformer.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

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

`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 `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

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.

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` 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.

FREQ

Center frequencies of subbands. `FREQ` is a column vector of length K, where K is the number of subbands in the `NumSubbands` property.

Examples

Apply subband phase shift beamformer to an 11-element ULA. The incident angle of the signal is 10 degrees in azimuth and 30 degrees in elevation.

```
% Signal simulation
ha = phased.ULA('NumElements',11,'ElementSpacing',0.3);
ha.Element.FrequencyRange = [20 20000];
fs = 1e3; carrierFreq = 2e3; t = (0:1/fs:2)';
x = chirp(t,0,2,fs);
c = 1500; % Wave propagation speed (m/s)
hc = phased.WidebandCollector('Sensor',ha,...
    'PropagationSpeed',c,'SampleRate',fs,...
    'ModulatedInput',true,'CarrierFrequency',carrierFreq);
incidentAngle = [10; 30];
x = step(hc,x,incidentAngle);
noise = 0.3*(randn(size(x)) + 1j*randn(size(x)));
rx = x+noise;

% Beamforming
hbf = phased.SubbandPhaseShiftBeamformer('SensorArray',ha,...
    'Direction',incidentAngle,...
    'OperatingFrequency',carrierFreq,'PropagationSpeed',c,...
    'SampleRate',fs,'SubbandsOutputPort',true,...
    'WeightsOutputPort',true);
[y,w,subbandfreq] = step(hbf,rx);
```

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`

Purpose

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 3-802.
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.

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.

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`

Methods

clone	Create ULA monopulse tracker object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform monopulse tracking using ULA

Examples

Determine the direction of a target at around 60 degrees broadside angle of a ULA.

```
ha = phased.ULA('NumElements',4);
hstv = phased.SteeringVector('SensorArray',ha);
hmp = phased.SumDifferenceMonopulseTracker('SensorArray',ha);
x = step(hstv,hmp.OperatingFrequency,60.1).';
est_dir = step(hmp,x,60);
```

Algorithms

The tracker uses a sum-and-difference monopulse algorithm to estimate the direction. The tracker obtains the difference steering vector by phase-reversing the latter half of the sum steering vector.

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

`phased.BeamscanEstimator` **|**
`phased.SumDifferenceMonopulseTracker2D` **|**

Purpose Create ULA monopulse tracker object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.SumDifferenceMonopulseTracker.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.SumDifferenceMonopulseTracker.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.SumDifferenceMonopulseTracker.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax `TF = isLocked(H)`

Description `TF = isLocked(H)` returns the locked status, `TF`, for the `SumDifferenceMonopulseTracker` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.SumDifferenceMonopulseTracker.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.SumDifferenceMonopulseTracker.step

Purpose

Perform monopulse tracking using ULA

Syntax

`ESTANG = step(H,X,STANG)`

Description

`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 `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

H

Tracker object of type `phased.SumDifferenceMonopulseTracker`.

X

Input signal, specified as a row vector whose number of columns corresponds to number of channels.

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`.

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

Determine the direction of a target at around 60 degrees broadside angle of a ULA.

```
ha = phased.ULA('NumElements',4);
hstv = phased.SteeringVector('SensorArray',ha);
hmp = phased.SumDifferenceMonopulseTracker('SensorArray',ha);
x = step(hstv,hmp.OperatingFrequency,60.1).';
est_dir = step(hmp,x,60);
```

Algorithms

The tracker uses a sum-and-difference monopulse algorithm to estimate the direction. The tracker obtains the difference steering vector by phase-reversing the latter half of the sum steering vector.

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

Purpose

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 3-812.
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.

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.

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`

Methods

clone	Create URA monopulse tracker object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform monopulse tracking using URA

Examples

Determine the direction of a target at around 60 degrees azimuth and 20 degrees elevation of a URA.

```
ha = phased.URA('Size',4);
hstv = phased.SteeringVector('SensorArray',ha);
hmp = phased.SumDifferenceMonopulseTracker2D('SensorArray',ha);
x = step(hstv,hmp.OperatingFrequency,[60.1; 19.5]).';
est_dir = step(hmp,x,[60; 20]);
```

Algorithms

The tracker uses a sum-and-difference monopulse algorithm to estimate the direction. The tracker obtains the difference steering vector by phase-reversing the latter half of the sum steering vector.

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

`phased.BeamscanEstimator` |
`phased.SumDifferenceMonopulseTracker` |

phased.SumDifferenceMonopulseTracker2D.clone

Purpose Create URA monopulse tracker object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.SumDifferenceMonopulseTracker2D.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.SumDifferenceMonopulseTracker2D.getNumOutpu

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.SumDifferenceMonopulseTracker2D.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `SumDifferenceMonopulseTracker2D` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.SumDifferenceMonopulseTracker2D.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.SumDifferenceMonopulseTracker2D.step

Purpose

Perform monopulse tracking using URA

Syntax

`ESTANG = step(H,X,STANG)`

Description

`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 `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

H

Tracker object of type `phased.SumDifferenceMonopulseTracker2D`.

X

Input signal, specified as a row vector whose number of columns corresponds to number of channels.

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`.

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

Determine the direction of a target at around 60 degrees azimuth and 20 degrees elevation of a URA.

```
ha = phased.URA('Size',4);
hstv = phased.SteeringVector('SensorArray',ha);
hmp = phased.SumDifferenceMonopulseTracker2D('SensorArray',ha);
x = step(hstv,hmp.OperatingFrequency,[60.1; 19.5]).';
est_dir = step(hmp,x,[60; 20]);
```

Algorithms

The tracker uses a sum-and-difference monopulse algorithm to estimate the direction. The tracker obtains the difference steering vector by phase-reversing the latter half of the sum steering vector.

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

Purpose

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 3-822.
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.

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.

Default: Speed of light

SampleRate

Signal sampling rate

Specify the signal sampling rate (in hertz) as a positive scalar.

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 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'`.

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

clone	Create time delay beamformer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform time delay beamforming

Examples

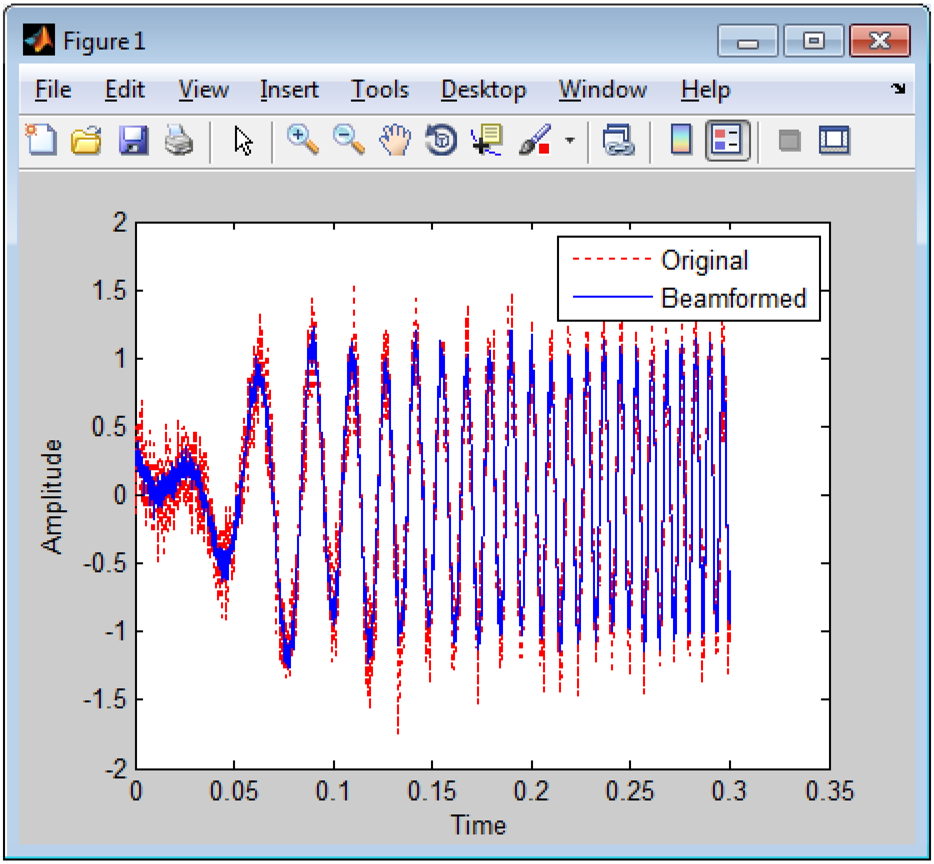
Apply a time delay beamformer to an 11-element array. The incident angle of the signal is –50 degrees in azimuth and 30 degrees in elevation.

```
% Signal simulation
ha = phased.ULA('NumElements',11,'ElementSpacing',0.04);
ha.Element.FrequencyRange = [20 20000];
fs = 8e3; t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340; % Wave propagation speed (m/s)
hc = phased.WidebandCollector('Sensor',ha,...
    'PropagationSpeed',c,'SampleRate',fs,'ModulatedInput',false);
incidentAngle = [-50;30];
x = step(hc,x.',incidentAngle);
```

```
noise = 0.2*randn(size(x));
rx = x+noise;

% Beamforming
hbf = phased.TimeDelayBeamformer('SensorArray',ha,...
    'SampleRate',fs,'PropagationSpeed',c,...
    'Direction',incidentAngle);
y = step(hbf,rx);

% Plot
plot(t,rx(:,6),'r:',t,y);
xlabel('Time'); ylabel('Amplitude');
legend('Original','Beamformed');
```



References

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

See Also

`phased.FrostBeamformer` | `phased.PhaseShiftBeamformer` | `phased.SubbandPhaseShiftBeamformer` | `phased.TimeDelayLCMVBeamformer` | `uv2azel` | `phitheta2azel`

Related Examples

- "Wideband Beamforming"

phased.TimeDelayBeamformer.clone

Purpose Create time delay beamformer object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.TimeDelayBeamformer.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.TimeDelayBeamformer.getNumOutputs

Purpose

Number of outputs from step method

Syntax

`N = getNumOutputs(H)`

Description

`N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `TimeDelayBeamformer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.TimeDelayBeamformer.release

Purpose

Allow property value and input characteristics changes

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Perform time delay beamforming

Syntax

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

Description

`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.

`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 `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.

Examples

Apply a time delay beamformer to an 11-element array. The incident angle of the signal is –50 degrees in azimuth and 30 degrees in elevation.

```
% Signal simulation
ha = phased.ULA('NumElements',11,'ElementSpacing',0.04);
ha.Element.FrequencyRange = [20 20000];
fs = 8e3; t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340; % Wave propagation speed (m/s)
hc = phased.WidebandCollector('Sensor',ha,...
    'PropagationSpeed',c,'SampleRate',fs,'ModulatedInput',false);
incidentAngle = [-50;30];
x = step(hc,x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x+noise;

% Beamforming
hbf = phased.TimeDelayBeamformer('SensorArray',ha,...
    'SampleRate',fs,'PropagationSpeed',c,...
    'Direction',incidentAngle);
y = step(hbf,rx);
```

See Also

`uv2azel` | `phitheta2azel`

phased.TimeDelayLCMVBeamformer

Purpose

Time delay LCMV beamformer

Description

The `TimeDelayLCMVBeamformer` object implements a time delay linear constraint minimum variance beamformer.

The `BeamscanEstimator` object calculates a beamscan spatial spectrum estimate for a uniform linear array.

To compute the beamformed signal:

1. Define and set up your time delay LCMV beamformer. See "Construction" on page 3-835.
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.

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.

Default: Speed of light

SampleRate

Signal sampling rate

Specify the signal sampling rate (in hertz) as a positive scalar.

Default: `1e6`

FilterLength

FIR filter length

Specify the length of the FIR filter behind each sensor element in the array as a positive integer.

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 degrees of freedom of the beamformer. For a time delay LCMV beamformer, `H`, M is given by `H.SensorArray*H.FilterLength`.

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.

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.

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 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'`.

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

clone	Create time delay LCMV beamformer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Perform time delay LCMV beamforming

Examples

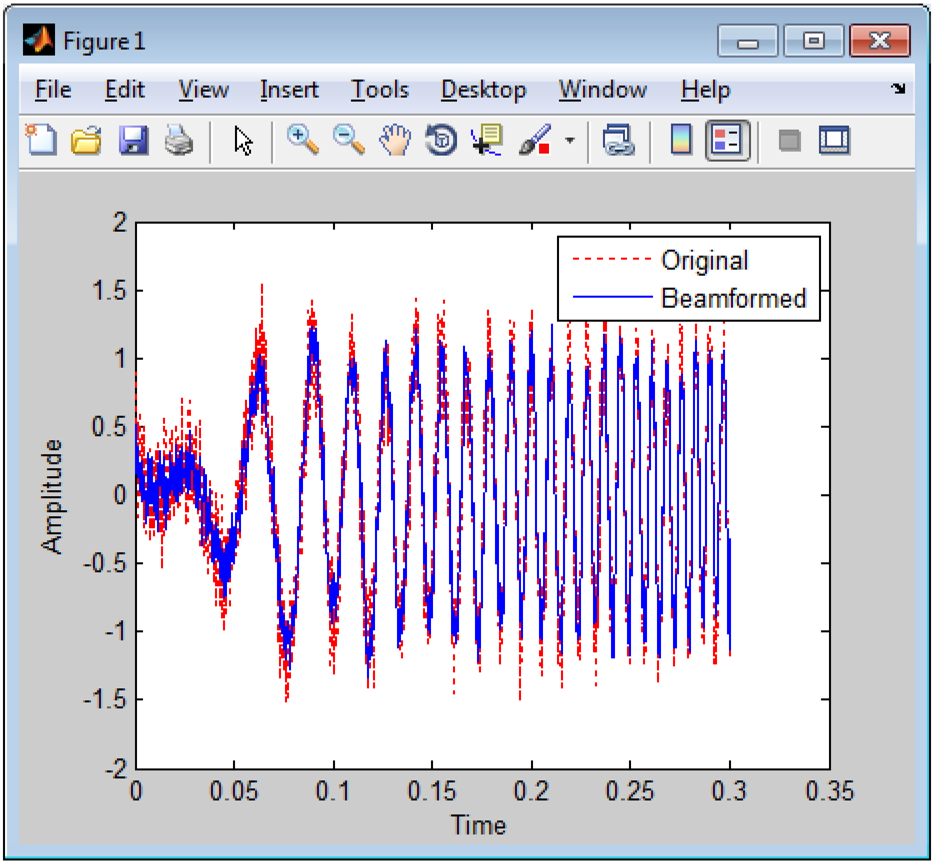
Apply a time delay LCMV beamformer to an 11-element array. The incident angle of the signal is –50 degrees in azimuth and 30 degrees in elevation.

```
% Signal simulation
ha = phased.ULA('NumElements',11,'ElementSpacing',0.04);
ha.Element.FrequencyRange = [20 20000];
fs = 8e3; t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340; % Wave propagation speed (m/s)
hc = phased.WidebandCollector('Sensor',ha,...
    'PropagationSpeed',c,'SampleRate',fs,'ModulatedInput',false);
incidentAngle = [-50; 30];
x = step(hc,x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x+noise;

% Beamforming
ha = phased.ULA('NumElements',11,'ElementSpacing',0.04);
```

```
hbf = phased.TimeDelayLCMVBeamformer('SensorArray',ha,...
    'PropagationSpeed',c,'SampleRate',fs,'FilterLength',5,...
    'Direction',incidentAngle);
hbf.Constraint = kron(eye(5),ones(11,1));
hbf.DesiredResponse = eye(5, 1);
y = step(hbf,rx);

% Plot
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.

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

`phased.FrostBeamformer` | `phased.PhaseShiftBeamformer` | `phased.SubbandPhaseShiftBeamformer` | `phased.TimeDelayBeamformer` | `uv2azel` | `phitheta2azel`

Related Examples

- “Wideband Beamforming”

Purpose

Create time delay LCMV beamformer object with same property values

Syntax

`C = clone(H)`

Description

`C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.TimeDelayLCMVBeamformer.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.TimeDelayLCMVBeamformer.getNumOutputs

Purpose

Number of outputs from step method

Syntax

`N = getNumOutputs(H)`

Description

`N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.TimeDelayLCMVBeamformer.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `TimeDelayLCMVBeamformer` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.TimeDelayLCMVBeamformer.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

phased.TimeDelayLCMVBeamformer.step

Purpose

Perform time delay LCMV beamforming

Syntax

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

Description

`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. `Y` is a column vector of length M. M must be larger than the FIR filter length specified in the `FilterLength` property.

`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 degrees, and the elevation angle must be between –90 and 90 degrees.

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 degrees of freedom of the beamformer. For a time delay LCMV beamformer, `H`, L is given by `H.SensorArray*H.FilterLength`.

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.

Examples

Apply a time delay LCMV beamformer to an 11-element array. The incident angle of the signal is –50 degrees in azimuth and 30 degrees in elevation.

```
% Signal simulation
ha = phased.ULA('NumElements',11,'ElementSpacing',0.04);
ha.Element.FrequencyRange = [20 20000];
fs = 8e3; t = 0:1/fs:0.3;
x = chirp(t,0,1,500);
c = 340; % Wave propagation speed (m/s)
hc = phased.WidebandCollector('Sensor',ha,...
    'PropagationSpeed',c,'SampleRate',fs,'ModulatedInput',false);
incidentAngle = [-50; 30];
x = step(hc,x.',incidentAngle);
noise = 0.2*randn(size(x));
rx = x+noise;

% Beamforming
ha = phased.ULA('NumElements',11,'ElementSpacing',0.04);
hbf = phased.TimeDelayLCMVBeamformer('SensorArray',ha,...
    'PropagationSpeed',c,'SampleRate',fs,'FilterLength',5,...
    'Direction',incidentAngle);
hbf.Constraint = kron(eye(5),ones(11,1));
hbf.DesiredResponse = eye(5, 1);
y = step(hbf,rx);
```

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`

Purpose

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 3-851.
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.

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

RangeLoss

Loss at each input sample range

Specify the loss (in decibels) due to the range for each sample in the input signal as a vector.

Default: `0`

ReferenceLoss

Loss at reference range

Specify the loss (in decibels) at a given reference range as a scalar.

Default: `0`

Methods

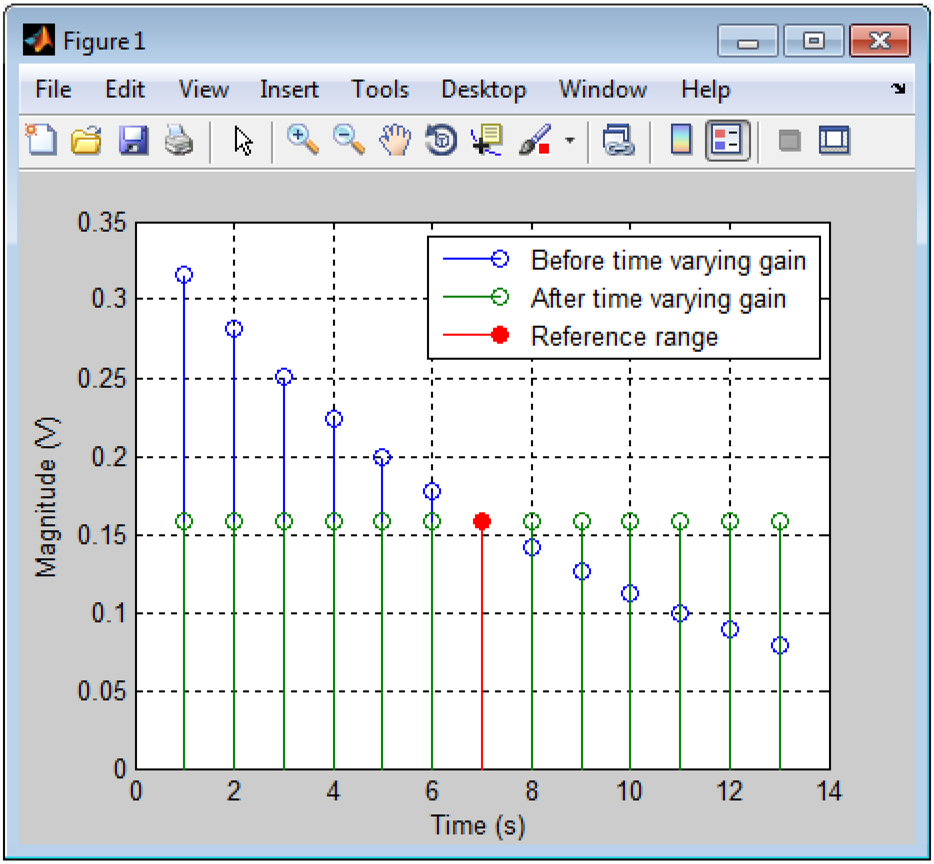
clone	Create time varying gain object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Apply time varying gains to input signal

Examples

Apply time varying gain to a signal to compensate for signal power loss due to range.

```
rngloss = 10:22; refloss = 16; % in dB
t = (1:length(rngloss))';
x = 1./db2mag(rngloss(:));
H = phased.TimeVaryingGain('RangeLoss',rngloss,...
    'ReferenceLoss',refloss);
y = step(H,x);

% Plot signals
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');
```



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.

See Also

`phased.MatchedFilter` | `pulsint`

phased.TimeVaryingGain.clone

Purpose Create time varying gain object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.TimeVaryingGain.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.TimeVaryingGain.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.TimeVaryingGain.isLocked

Purpose Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description `TF = isLocked(H)` returns the locked status, `TF` of the `TimeVaryingGain` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.TimeVaryingGain.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose

Apply time varying gains to input signal

Syntax

`Y = step(H,X)`

Description

`Y = step(H,X)` applies time varying gains to the input signal `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` must match the length of the loss vector specified in the `RangeLoss` property. `Y` has the same dimensionality as `X`.

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.

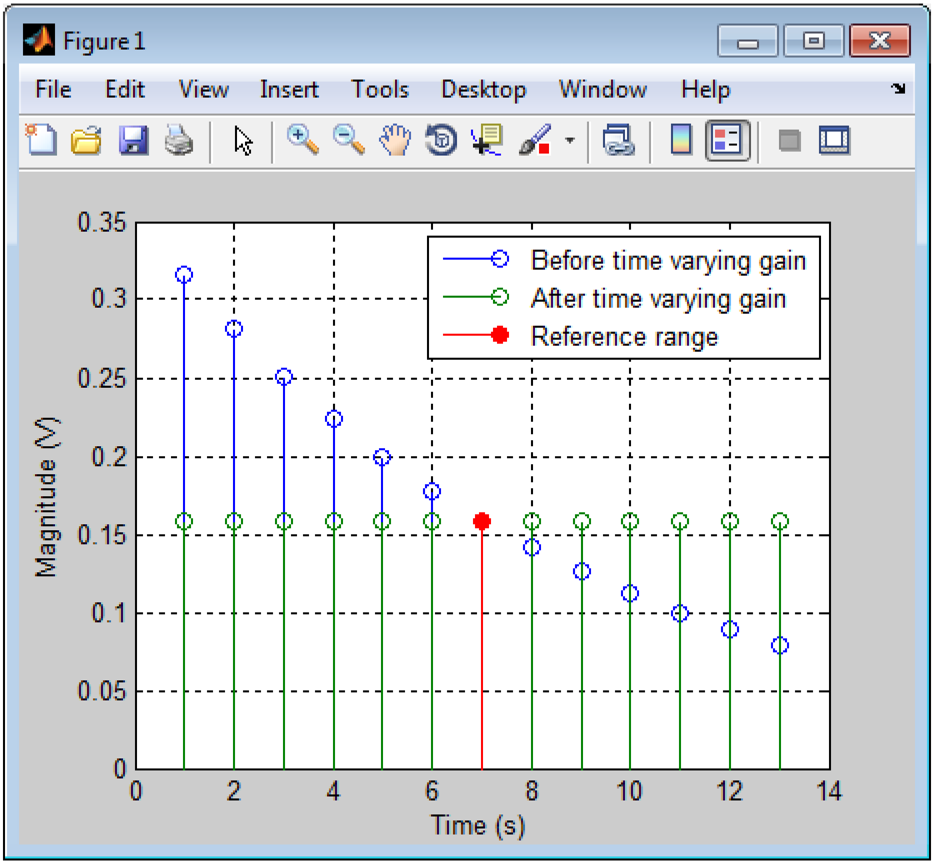
Examples

Apply time varying gain to a signal to compensate for signal power loss due to range.

```
rngloss = 10:22; refloss = 16; % in dB
t = (1:length(rngloss))';
x = 1./db2mag(rngloss(:));
H = phased.TimeVaryingGain('RangeLoss',rngloss,...
    'ReferenceLoss',refloss);
y = step(H,x);

% Plot signals
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');
```



Purpose

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 3-861.
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.

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 $e^{j\varphi}$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

clone	Create transmitter object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset	Reset states of transmitter object
step	Transmit pulses

Examples

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;
hwav = phased.LinearFMWaveform('SampleRate',fs,...
    'PulseWidth',1e-5,'SweepBandwidth',5e6);
x = step(hwav);
htx = phased.Transmitter('PeakPower',5e3);
y = step(htx,x);
```

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.

See Also

`phased.Radiator` | `phased.ReceiverPreamp` |

phased.Transmitter.clone

Purpose Create transmitter object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.Transmitter.getNumInputs

Purpose

Number of expected inputs to step method

Syntax

`N = getNumInputs(H)`

Description

`N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.Transmitter.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `Transmitter` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.Transmitter.release

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles, or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

Purpose 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'`.

phased.Transmitter.step

Purpose

Transmit pulses

Syntax

```
Y = step(H,X)
[Y,STATUS] = step(H,X)
[Y,PHNOISE] = step(H,X)
```

Description

`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.

`[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 `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.

Examples

Transmit a pulse containing a linear FM waveform. The sample rate is 10 MHz and the pulse repetition frequency is 50 kHz. The transmitter peak power is 5 kw.

```
fs = 1e7;
hwav = phased.LinearFMWaveform('SampleRate',fs,...
    'PulseWidth',1e-5,'SweepBandwidth',5e6);
x = step(hwav);
htx = phased.Transmitter('PeakPower',5e3);
y = step(htx,x);
```

phased.ULA

Purpose

Uniform linear array

Description

The `ULA` object creates a uniform linear array.

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 3-874.

2. Call `step` to compute the response according to the properties of `phased.ULA`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.ULA` creates a uniform linear array (ULA) System object, `H`. The object models a ULA formed with identical 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.ULA(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.ULA(N,D,Name,Value)` creates a ULA object, `H`, 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. To specify a value-only argument, you must also specify all preceding value-only arguments. You can specify name-value pair arguments in any order.

Properties

Element

Element of array

Specify the element of the sensor array as a handle. The element must be an element object in the `phased` package.

Default: An isotropic antenna element that operates between 300 MHz and 1 GHz

NumElements

Number of elements

An integer containing the number of elements in the array.

Default: `2`

ElementSpacing

Element spacing

A scalar containing the spacing (in meters) between two adjacent elements in the array.

Default: `0.5`

Methods

clone	Create ULA object with same property values
collectPlaneWave	Simulate received plane waves
getElementPosition	Positions of array elements
getNumElements	Number of elements in array
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of array
release	Allow property value and input characteristics

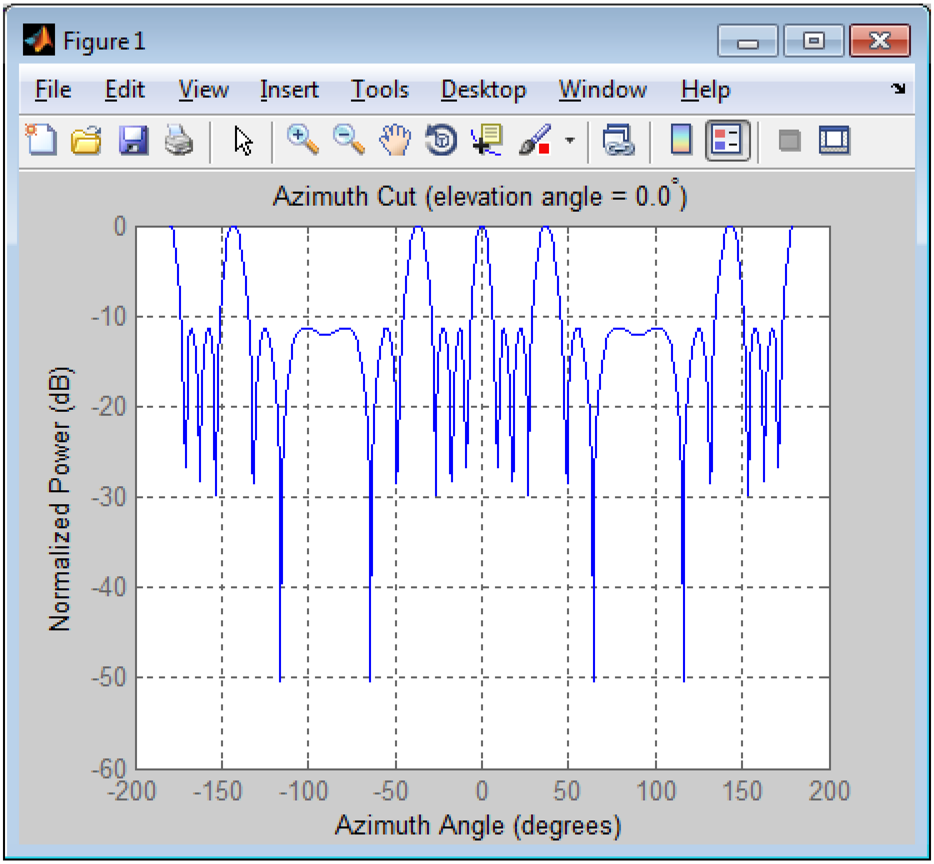
step	Output responses of array elements
viewArray	View array geometry

Examples

Response of Antenna Array

Create a 4-element ULA and find the response of each element at the 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');
plotResponse(ha,fc,c)
```

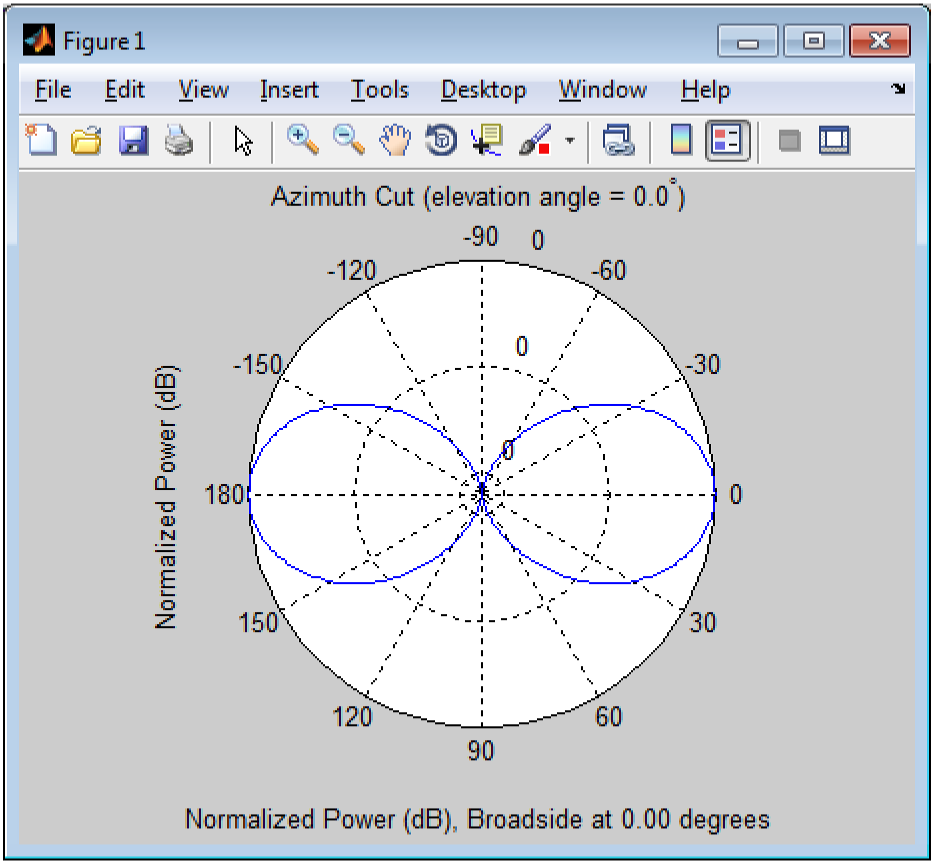


Response of Microphone Array

Find and plot the response of an array of 10 microphones. In this example, the `Element` property matches the acoustic frequency range of a microphone.

```
hmic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
Nele = 10;
hula = phased.ULA('NumElements',Nele,...
```

```
    'ElementSpacing',3e-3,...
    'Element',hmic);
fc = 100;
ang = [0; 0];
resp = step(hula,fc,ang);
c = 340;
plotResponse(hula,fc,c,'RespCut','Az','Format','Polar');
```



References

[1] Brookner, E., ed. *Radar Technology*. Lexington, MA: LexBook, 1996.

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

See Also

`phased.ReplicatedSubarray` **|** `phased.PartitionedArray` **|** `phased.ConformalArray` **|** `phased.CosineAntennaElement` **|** `phased.CustomAntennaElement` **|** `phased.IsotropicAntennaElement` **|** `phased.URA` **|**

Related Examples

- Phased Array Gallery

phased.ULA.clone

Purpose Create ULA object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

Purpose

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)` uses `FREQ` as the incoming signal's carrier frequency.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)` uses `C` as the signal's propagation speed. `C` must be a scalar.

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 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, 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 the received signal at a 4-element ULA.

The signals arrive from 10 degrees 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.

```
ha = phased.ULA(4);
y = collectPlaneWave(ha,randn(4,2),[10 30],1e8,...
    physconst('LightSpeed'));
```

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`

phased.ULA.getElementPosition

Purpose

Positions of array elements

Syntax

```
POS = getElementPosition(H)
```

Description

`POS = getElementPosition(H)` returns the element positions of the ULA, `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]. 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.

Examples

Construct a default ULA, and obtain the element positions.

```
ha = phased.ULA;
pos = getElementPosition(ha)
```

phased.ULA.getNumElements

Purpose

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

Construct a default ULA, and obtain the number of elements in that array.

```
ha = phased.ULA;
N = getNumElements(ha)
```

phased.ULA.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.ULA.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `ULA` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.ULA.plotResponse

Purpose

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.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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can

specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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'`, and `'pow'`.

Default: `'db'`

Weights

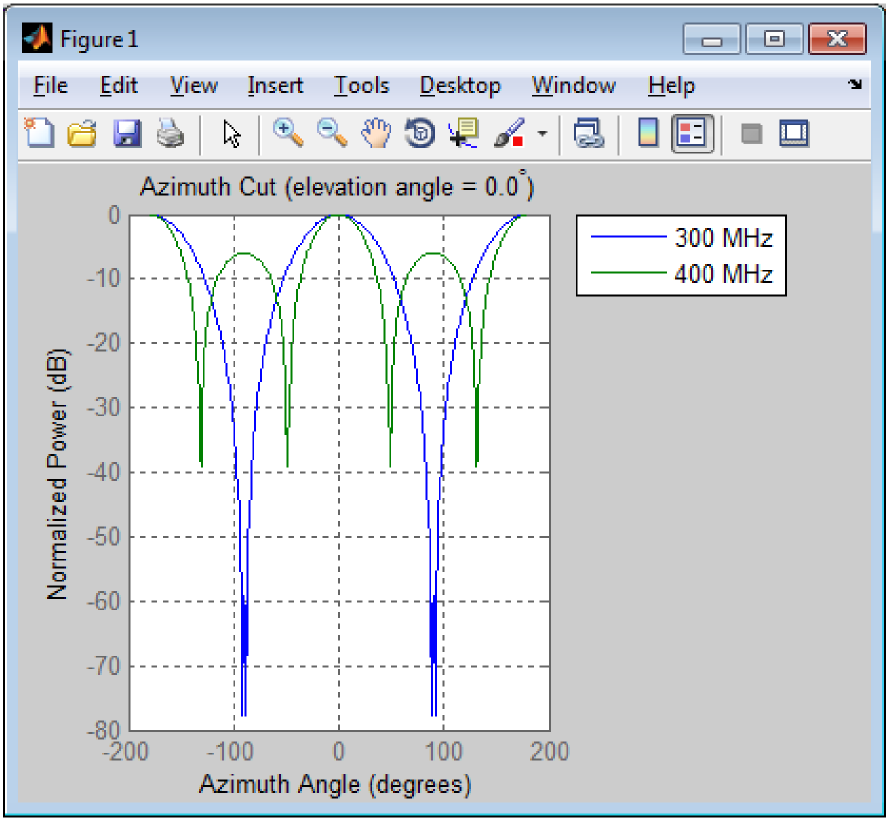
Weights applied to the array, specified as a length-*N* column vector or *N*-by-*M* matrix. *N* is the number of elements in the array. *M* is the number of frequencies in `FREQ`. If `Weights` is a vector, the function applies the same weights to each frequency. If `Weights` is a matrix, the function applies each column of weight values to the corresponding frequency in `FREQ`.

Examples

Line Plot Showing Multiple Frequencies

Plot the azimuth cut response of a uniform linear array along 0 elevation using a line plot. The plot shows the responses at operating frequencies of 300 MHz and 400 MHz.

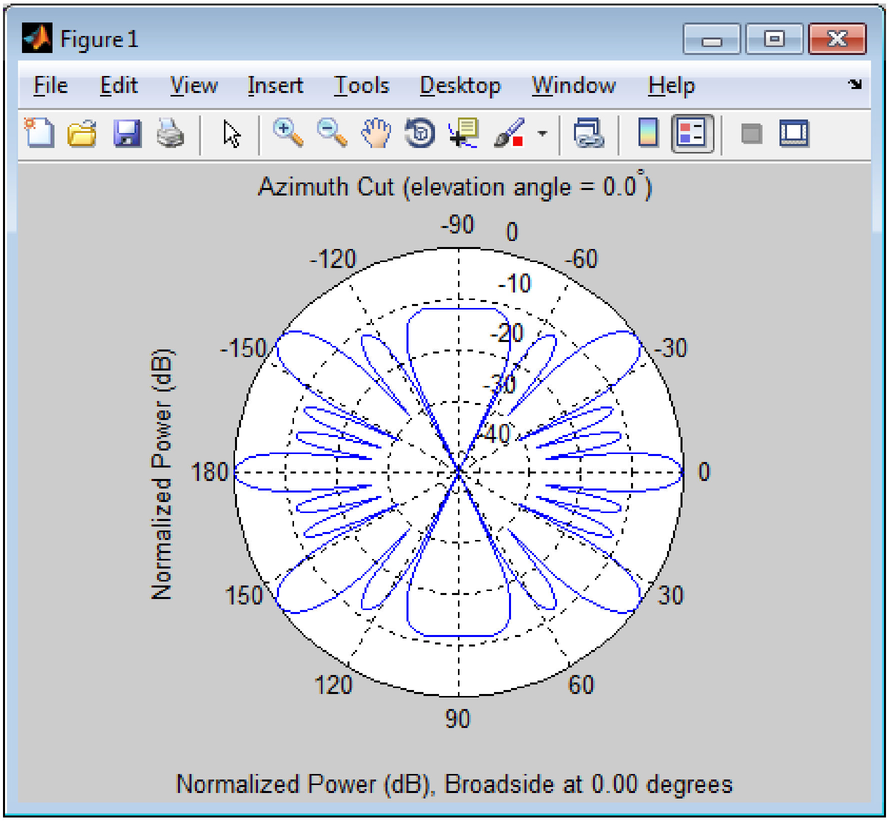
```
h = phased.ULA;
fc = [3e8 4e8];
c = physconst('LightSpeed');
plotResponse(h,fc,c)
```



Polar Plot

Construct a 4-element ULA and plot its azimuth response in polar format. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

```
ha = phased.ULA(4);
fc = 1e9; c = 3e8;
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



See Also

`uv2azel` | `azel2uv`

Purpose

Allow property value and input characteristics

Syntax

`release(H)`

Description

`release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.ULA.step

Purpose

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

`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 `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

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` 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`

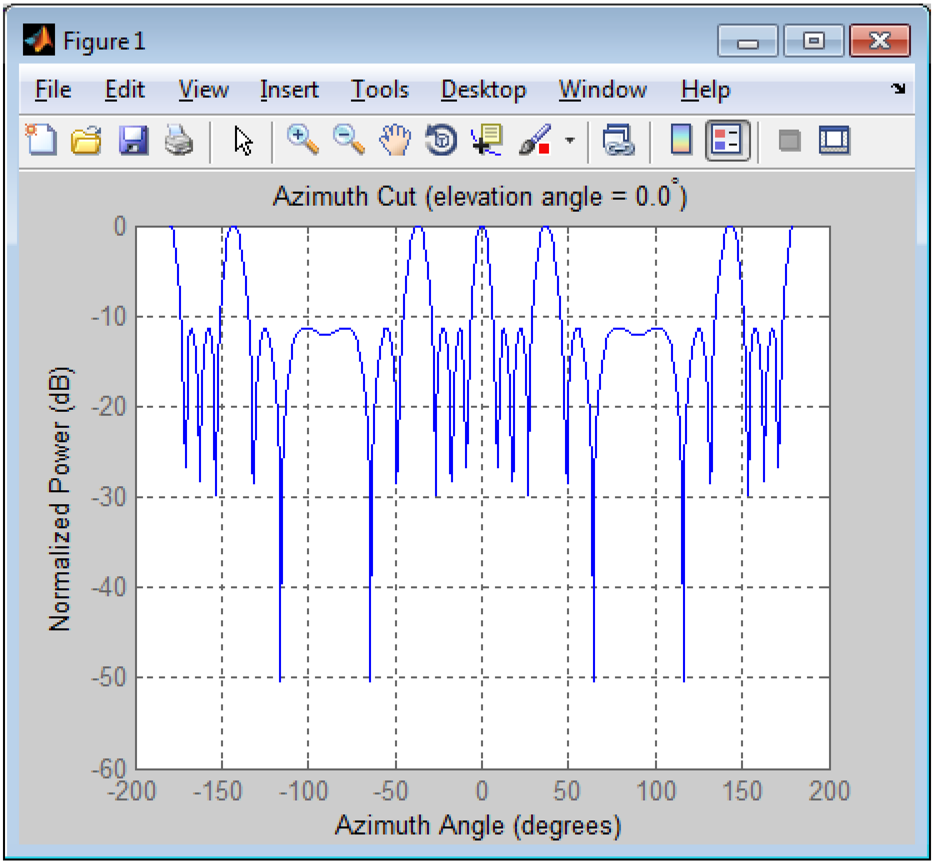
Responses of array elements. `RESP` has dimensions N-by-M-by-L. N is the number of elements in the phased array. Each column of `RESP` contains the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.

Examples

Response of Antenna Array

Create a 4-element ULA and find the response of each element at the 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');
plotResponse(ha,fc,c)
```

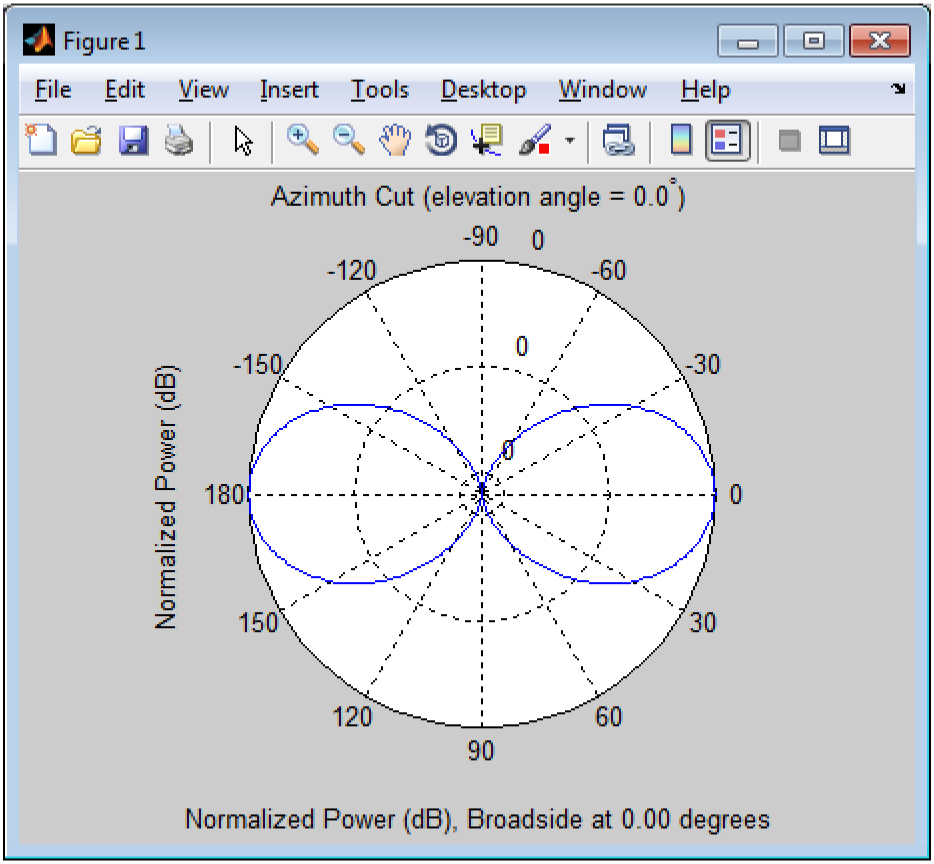


Response of Microphone Array

Find and plot the response of an array of 10 microphones. In this example, the `Element` property matches the acoustic frequency range of a microphone.

```
hmic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
Nele = 10;
hula = phased.ULA('NumElements',Nele,...
```

```
    'ElementSpacing',3e-3,...
    'Element',hmic);
fc = 100;
ang = [0; 0];
resp = step(hula,fc,ang);
c = 340;
plotResponse(hula,fc,c,'RespCut','Az','Format','Polar');
```



See Also

`uv2azel` | `phitheta2azel`

phased.ULA.viewArray

Purpose

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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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 string `'All'` to show indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

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`

Title

String specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

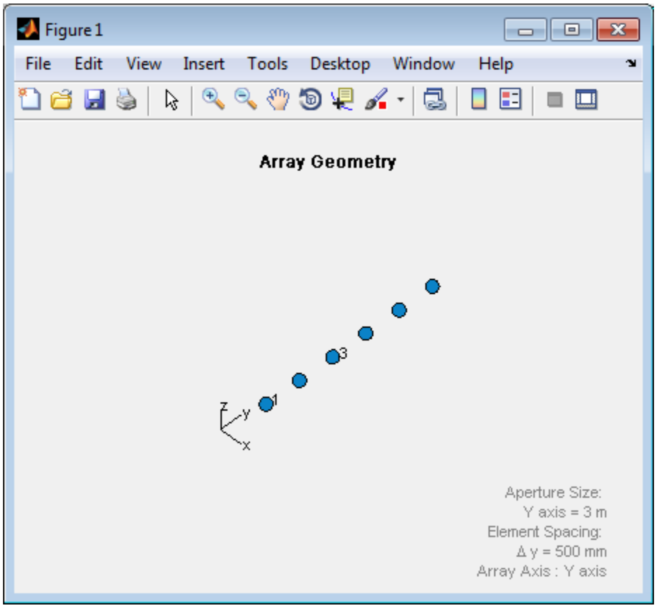
Handle of array elements in figure window.

Examples

Geometry and Indices of ULA Elements

Display the geometry of a 6-element ULA, and show the indices for the first and third elements.

```
ha = phased.ULA(6);
viewArray(ha,'ShowIndex',[1 3]);
```



Aperture Size:
Y axis = 3 m
Element Spacing:
Δ y = 500 mm
Array Axis : Y axis

See Also `phased.ArrayResponse` |

Related Examples

- Phased Array Gallery

Purpose

Uniform rectangular array

Description

The `URA` object constructs a uniform rectangular array.

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 3-901.
2. Call `step` to compute the response according to the properties of `phased.URA`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.URA` creates a uniform rectangular array (URA) System object, `H`. The object models a URA formed with identical sensor elements. Array elements are distributed in the *yz*-plane in a rectangular lattice. The array look direction is along the positive *x*-axis.

`H = phased.URA(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.URA(SZ,D,Name,Value)` creates a URA object,`H`, with the `Size` property set to `SZ`, the `ElementSpacing` property set to `D` and other specified property Names set to the specified Values. `SZ` and `D` are value-only arguments. To specify a value-only argument, you must also specify all preceding value-only arguments. You can specify name-value pair arguments in any order.

Properties

Element

Element of array

Specify the element of the sensor array as a handle. The element must be an element object in the `phased` package.

Default: An isotropic antenna element that operates between 300 MHz and 1 GHz

Size

Size of array

A 1-by-2 integer vector or an integer containing the size of the array. If `Size` is a 1-by-2 vector, the vector has the form `[NumberOfElementsInEachRow NumberOfElementInEachColumn]`. If `Size` is a scalar, the array has the same number of elements in each row and column.

Default: `[2 2]`

ElementSpacing

Element spacing

A 1-by-2 vector or a scalar containing the element spacing (in meters) of the array. If `ElementSpacing` is a 1x2 vector, it is in the form of `[SpacingAlongRow SpacingAlongColumn]`. If `ElementSpacing` is a scalar, the spacing along the row and the spacing along the column 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 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'`

Methods

clone	Create URA object with same property values
collectPlaneWave	Simulate received plane waves
getElementPosition	Positions of array elements
getNumElements	Number of elements in array
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotResponse	Plot response pattern of array
release	Allow property value and input characteristics
step	Output responses of array elements
viewArray	View array geometry

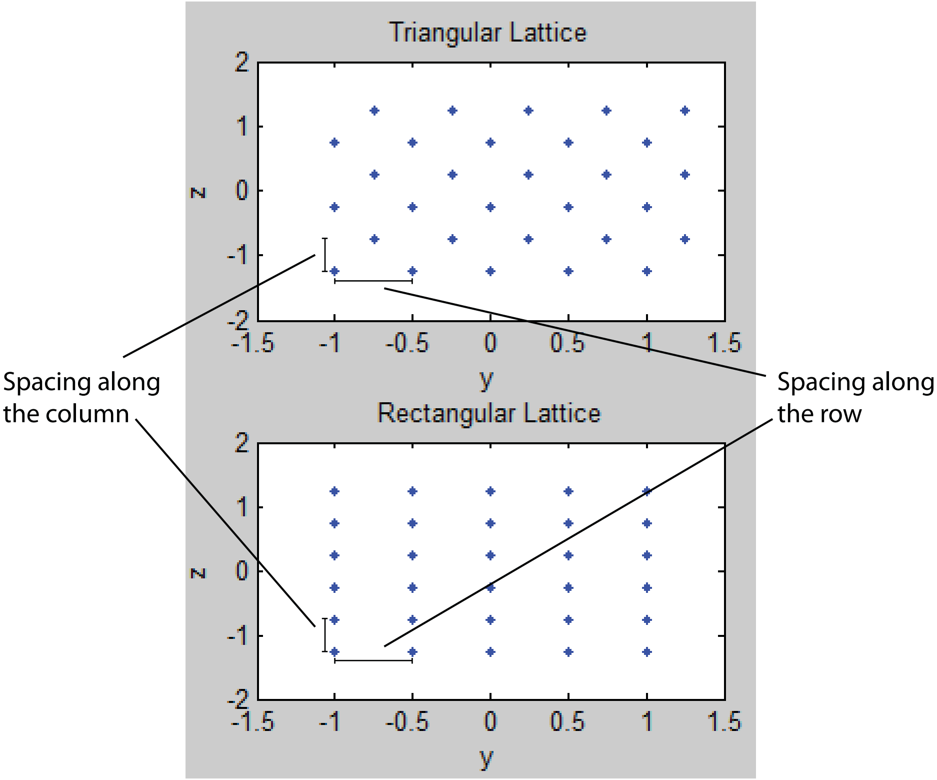
Definitions

Spacing Along the Row

The spacing along the row is the distance between adjacent elements in the same row.

Spacing Along the Column

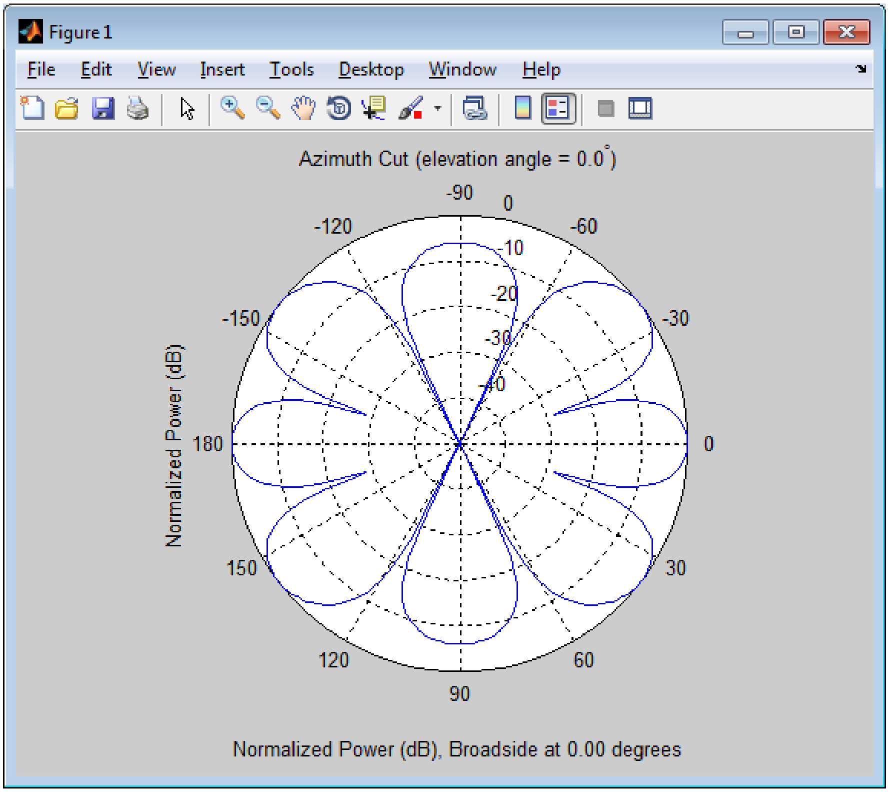
The spacing along the column is the distance in the column axis direction between adjacent rows.



Examples

Construct a 2-by-3 URA with a rectangular lattice, and find the response of each element at the boresight. Assume the operating frequency is 1 GHz. Finally, plot the azimuth response of the array.

```
ha = phased.URA('Size',[2 3]);
fc = 1e9; ang = [0;0];
resp = step(ha,fc,ang);
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



Comparison of Triangular and Rectangular Lattice

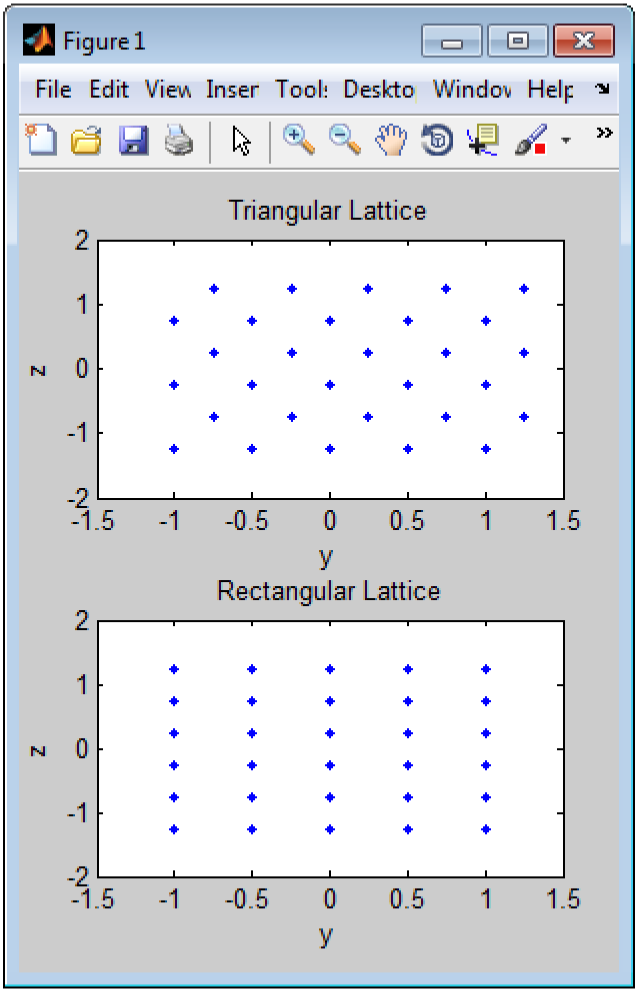
Find and plot the positions of the elements in a URA with a triangular lattice and a URA with a rectangular lattice. The element spacing is 0.5 for both lattices.

```
% Create URAs with triangular and rectangular lattices.
h_tri = phased.URA('Size',[5 6],'Lattice','Triangular');
h_rec = phased.URA('Size',[5 6],'Lattice','Rectangular');

% Get element positions for each array.
pos_tri = getElementPosition(h_tri);
```

```
pos_rec = getElementPosition(h_rec);
% Get y and z coordinates. All the x coordinates are zero.
pos_yz_tri = pos_tri(2:3,:);
pos_yz_rec = pos_rec(2:3,:);

% Plot element positions in yz-plane.
figure;
set(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')
```



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

See Also

`phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` |

Related Examples

- Phased Array Gallery

Purpose Create URA object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.URA.collectPlaneWave

Purpose

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)` uses `FREQ` as the incoming signal's carrier frequency.

`Y = collectPlaneWave(H,X,ANG,FREQ,C)` uses `C` as the signal's propagation speed. `C` must be a scalar.

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 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, 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 the received signal 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 degrees 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.

```
hURA = phased.URA([2 3]);
y = collectPlaneWave(hURA,randn(4,2),[10 30],1e8,...
        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`

Purpose

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 through each row, one after another. 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 10.

Examples

Construct a default URA with a rectangular lattice, and obtain the element positions.

```
ha = phased.URA;
pos = getElementPosition(ha)
```

phased.URA.getNumElements

Purpose 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 Construct a default URA, and obtain the number of elements.

```
ha = phased.URA;
N = getNumElements(ha)
```

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

phased.URA.getNumOutputs

Purpose Number of outputs from step method

Syntax `N = getNumOutputs(H)`

Description `N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

Purpose

Locked status for input attributes and nontunable properties

Syntax

`TF = isLocked(H)`

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `URA` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

phased.URA.plotResponse

Purpose

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.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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can

specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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.

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`

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'`, and `'pow'`.

Default: `'db'`

Weights

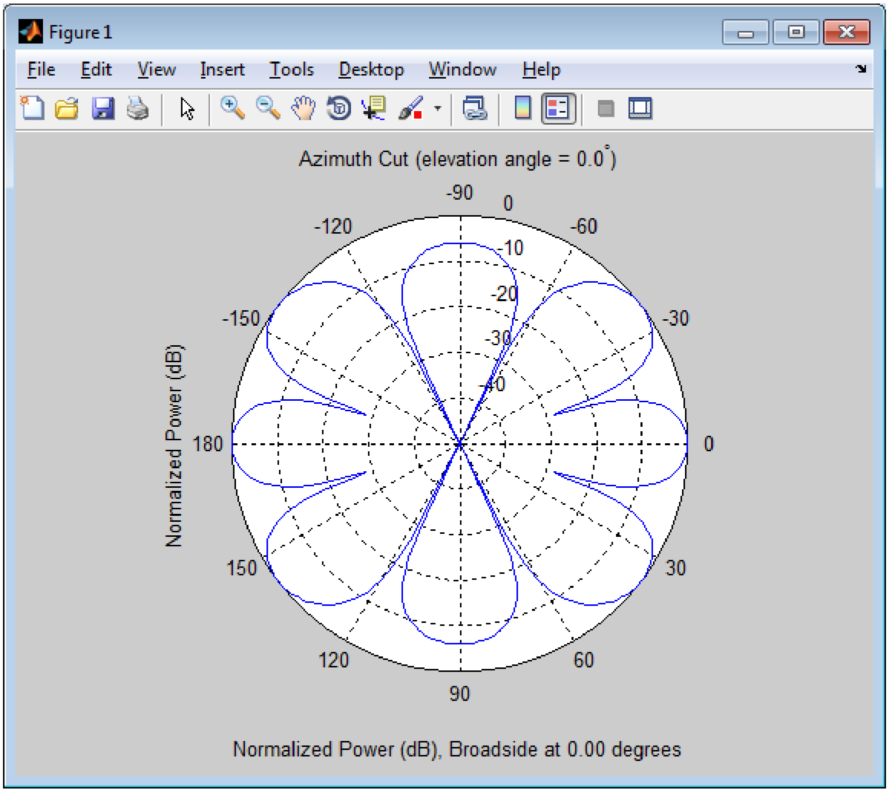
Weights applied to the array, specified as a length-N column vector or N-by-M matrix. N is the number of elements in the array. M is the number of frequencies in `FREQ`. If `Weights` is a vector, the function applies the same weights to each frequency. If `Weights` is a matrix, the function applies each column of weight values to the corresponding frequency in `FREQ`.

Examples

Azimuth Response of URA

Construct a 2-by-3 URA with a rectangular lattice, and plot that array's azimuth response.

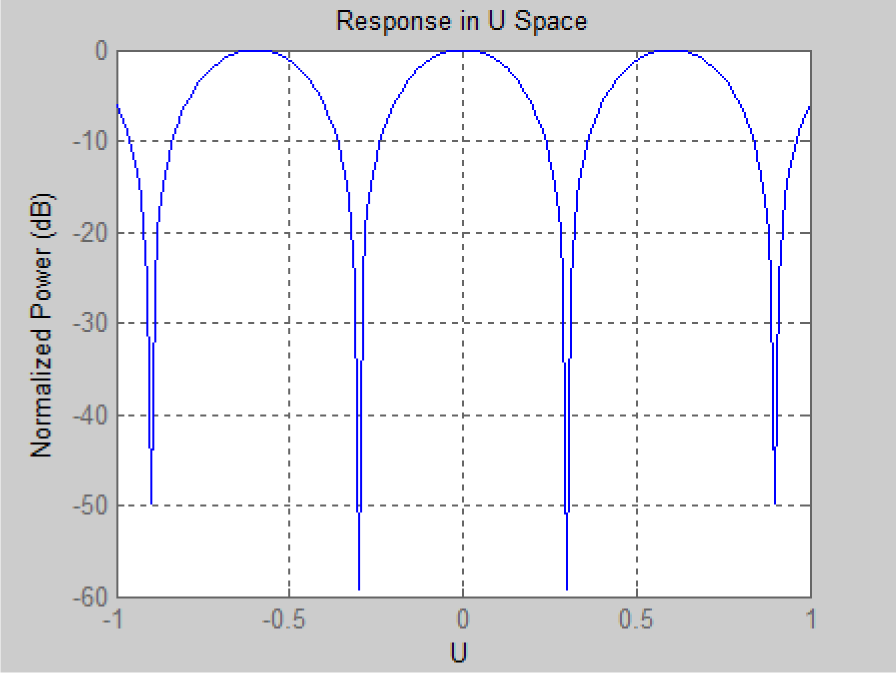
```
ha = phased.URA('Size',[2 3]);
fc = 1e9;
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



Array Response in U/V Space

Construct a 2-by-3 URA with a rectangular lattice. Plot the *u* cut of that array's response in *u*/*v* space.

```
ha = phased.URA('Size',[2 3]);
c = physconst('lightspeed');
plotResponse(ha,1e9,c,'Format','UV');
```



See Also `uv2azel` | `azel2uv`

Purpose Allow property value and input characteristics

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.URA.step

Purpose

Output responses of array elements

Syntax

`RESP = step(H,FREQ,ANG)`

Description

`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 `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

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` 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

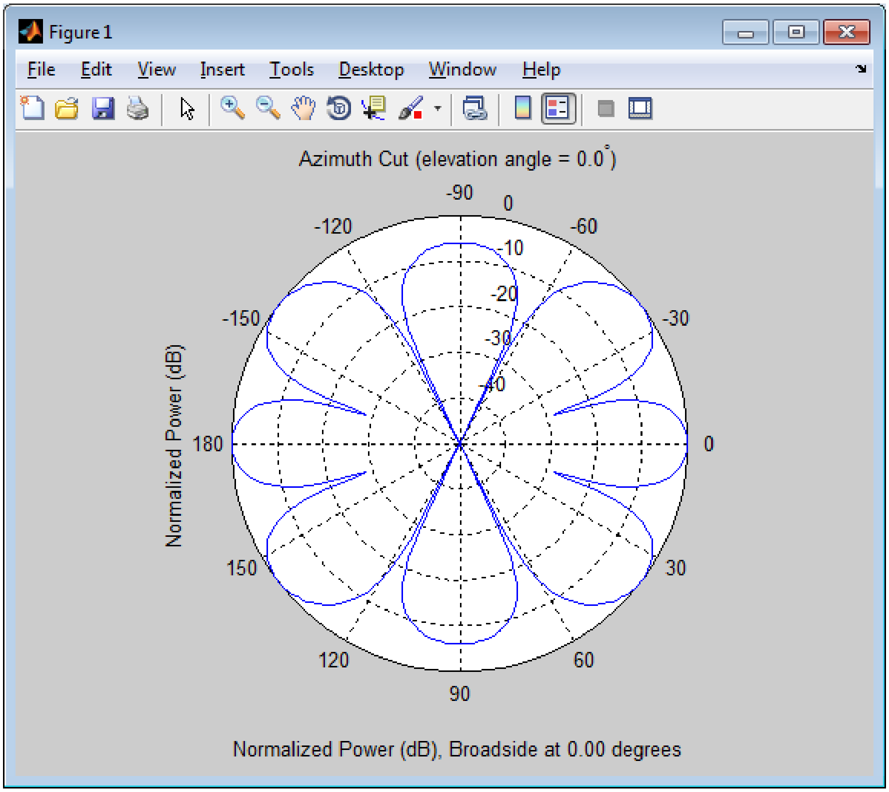
RESP

Responses of array elements. `RESP` has dimensions N-by-M-by-L. N is the number of elements in the phased array. 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

Construct a 2-by-3 URA with a rectangular lattice, and find the response of each element at the boresight. Assume the operating frequency is 1 GHz. Finally, plot the azimuth response of the array.

```
ha = phased.URA('Size',[2 3]);
fc = 1e9; ang = [0;0];
resp = step(ha,fc,ang);
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



See Also

`uv2azel` | `phitheta2azel`

Purpose

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 comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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 string `'All'` to show indices of all elements of the array or `'None'` to suppress indices.

Default: `'None'`

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`

Title

String specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

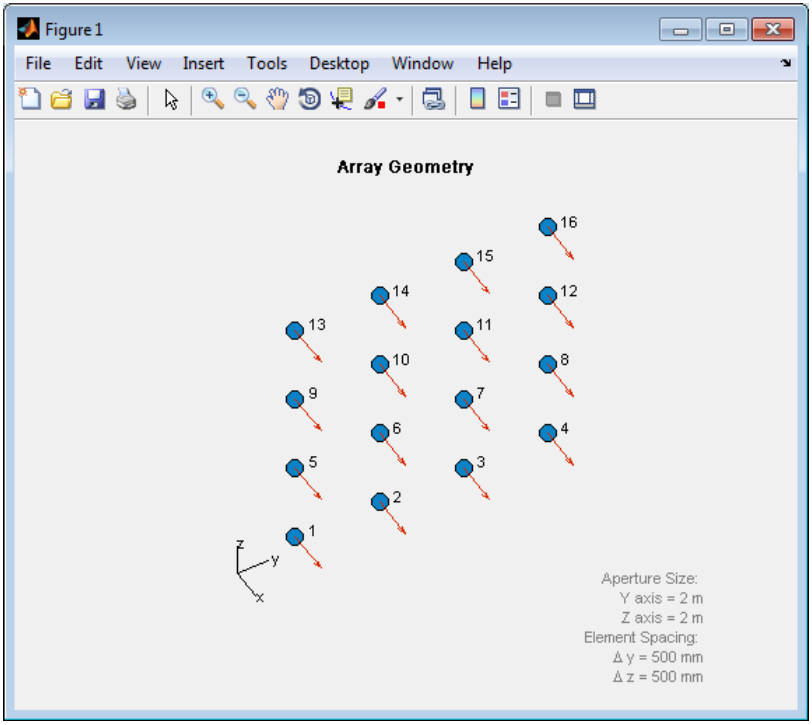
Handle of array elements in figure window.

Examples

Geometry, Normal Directions, and Indices of URA Elements

Display the element positions, normal directions, and indices for all elements of a 4-by-4 URA.

```
ha = phased.URA(4);
viewArray(ha,'ShowNormals',true,'ShowIndex','All');
```



See Also `phased.ArrayResponse` |

Related Examples

- Phased Array Gallery

phased.WidebandCollector

Purpose

Wideband signal collector

Description

The `WidebandCollector` object implements a wideband signal collector.

To compute the collected signal at the sensor(s):

1. Define and set up your wideband signal collector. See "Construction" on page 3-930.
2. Call `step` to collect the signal according to the properties of `phased.WidebandCollector`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.WidebandCollector` creates a wideband signal collector System object, `H`. The object collects incident wideband signals from given directions using a sensor array or a single element.

`H = phased.WidebandCollector(Name,Value)` creates a wideband signal collector 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

Sensor

Handle of sensor

Specify the sensor as a sensor array object or an element object in the `phased` package. If the sensor is an array, it 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

SampleRate

Sample rate

Specify the sample rate, in hertz, as a positive scalar. The default value corresponds to 1 MHz.

Default: `1e6`

ModulatedInput

Assume modulated input

Set this property to `true` to indicate the input signal is demodulated at a carrier frequency.

Default: `true`

CarrierFrequency

Carrier frequency

Specify the carrier frequency (in hertz) as a positive scalar. The default value of this property corresponds to 1 GHz. This property applies when the `ModulatedInput` property is `true`.

Default: `1e9`

WeightsInputPort

Enable weights input

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`

Wavefront

Type of incoming wavefront

Specify the type of incoming wavefront as one of `'Plane'`, or `'Unspecified'`:

- If you set the `Wavefront` property to `'Plane'`, the input signals are multiple plane waves impinging on the entire array. Each plane wave is received by all collecting elements. If the `Sensor` property is an array that contains subarrays, the `Wavefront` property must be `'Plane'`.
- If you set the `Wavefront` property to `'Unspecified'`, the input signals are individual waves impinging on individual sensors.

Default: `'Plane'`

Methods

clone	Create wideband collector object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
step	Collect signals

Examples

Collect signal with a single antenna.

```
ha = phased.IsotropicAntennaElement;
hc = phased.WidebandCollector('Sensor',ha);
x = [1;1];
incidentAngle = [10 30]';
y = step(hc,x,incidentAngle);
```

Collect a far field signal with a 5-element array.

```
ha = phased.ULA('NumElements',5);
hc = phased.WidebandCollector('Sensor',ha);
x = [1;1];
incidentAngle = [10 30]';
y = step(hc,x,incidentAngle);
```

Collect signal with a 3-element array. Each antenna collects a separate input signal from a separate direction.

```
ha = phased.ULA('NumElements',3);
hc = phased.WidebandCollector('Sensor',ha,...
    'Wavefront','Unspecified');
x = rand(10,3); % Each column is a signal for one element
incidentAngle = [10 0; 20 5; 45 2]'; % 3 angles for 3 signals
y = step(hc,x,incidentAngle);
```

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'`, `phased.WidebandCollector` collects each channel independently.

For further details, see [1].

References

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

See Also

`phased.Collector` |

phased.WidebandCollector.clone

Purpose Create wideband collector object with same property values

Syntax `C = clone(H)`

Description `C = clone(H)` creates an object, `C`, having the same property values and same states as `H`. If `H` is locked, so is `C`.

phased.WidebandCollector.getNumInputs

Purpose Number of expected inputs to step method

Syntax `N = getNumInputs(H)`

Description `N = getNumInputs(H)` returns a positive integer, `N`, representing the number of inputs (not counting the object itself) you must use when calling the `step` method. This value will change if you alter any properties that turn inputs on or off.

Purpose

Number of outputs from step method

Syntax

```
N = getNumOutputs(H)
```

Description

`N = getNumOutputs(H)` returns the number of outputs, `N`, from the `step` method. This value will change if you change any properties that turn outputs on or off.

phased.WidebandCollector.isLocked

Purpose

Locked status for input attributes and nontunable properties

Syntax

```
TF = isLocked(H)
```

Description

`TF = isLocked(H)` returns the locked status, `TF`, for the `WidebandCollector` System object.

The `isLocked` method returns a logical value that indicates whether input attributes and nontunable properties for the object are locked. The object performs an internal 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. After locking, the `isLocked` method returns a `true` value.

Purpose Allow property value and input characteristics changes

Syntax `release(H)`

Description `release(H)` releases system resources (such as memory, file handles or hardware connections) and allows all properties and input characteristics to be changed.

Note You can use the `release` method on a System object in code generated from MATLAB, but once you release its resources, you cannot use that System object again.

---

phased.WidebandCollector.step

Purpose

Collect signals

Syntax

```
Y = step(H,X,ANG)
Y = step(H,X,ANG,WEIGHTS)
Y = step(H,X,ANG,STEERANGLE)
Y = step(H,X,ANG,WEIGHTS,STEERANGLE)
```

Description

`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,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,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 `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

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.

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.

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 signal with a single antenna.

```
ha = phased.IsotropicAntennaElement;
hc = phased.WidebandCollector('Sensor',ha);
x = [1;1];
incidentAngle = [10 30]';
y = step(hc,x,incidentAngle);
```

Collect a far field signal with a 5-element array.

```
ha = phased.ULA('NumElements',5);
hc = phased.WidebandCollector('Sensor',ha);
x = [1;1];
incidentAngle = [10 30]';
y = step(hc,x,incidentAngle);
```

Collect signal with a 3-element array. Each antenna collects a separate input signal from a separate direction.

```
ha = phased.ULA('NumElements',3);
hc = phased.WidebandCollector('Sensor',ha,...
    'Wavefront','Unspecified');
x = rand(10,3); % Each column is a signal for one element
incidentAngle = [10 0; 20 5; 45 2]'; % 3 angles for 3 signals
y = step(hc,x,incidentAngle);
```

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'`, `phased.WidebandCollector` collects each channel independently.

For further details, see [1].

References

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

Functions-Alphabetical List

albersheim

Purpose

Required SNR using Albersheim's equation

Syntax

```
SNR = albersheim(prob_Detection,prob_FalseAlarm)
SNR = albersheim(prob_Detection,prob_FalseAlarm,N)
```

Description

`SNR = albersheim(prob_Detection,prob_FalseAlarm)` returns the signal-to-noise ratio in decibels. This value indicates the ratio required to achieve the given probabilities of detection `prob_Detection` and false alarm `prob_FalseAlarm` for a single sample.

`SNR = albersheim(prob_Detection,prob_FalseAlarm,N)` determines the required SNR for the noncoherent integration of `N` samples.

Definitions

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.

Albersheim's equation for the required SNR in decibels is:

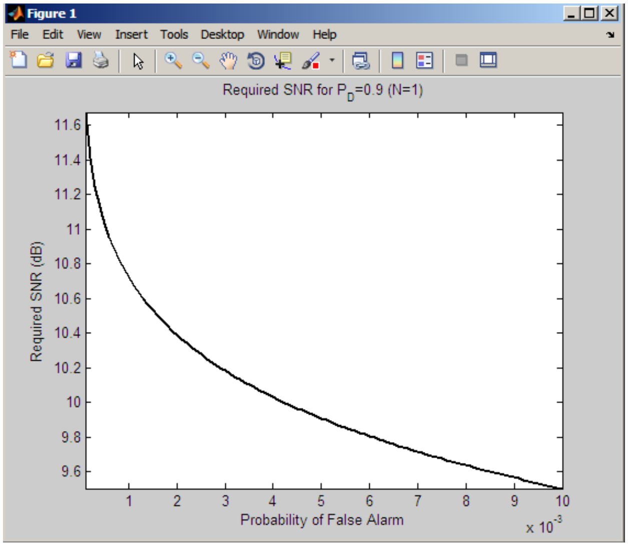
$$\mathrm{SNR} = -5\log_{10} N + [6.2 + 4.54 / \sqrt{N + 0.44}]\log_{10}(A + 0.12AB + 1.7B)$$

where `N` is the number of noncoherently integrated samples.

Examples

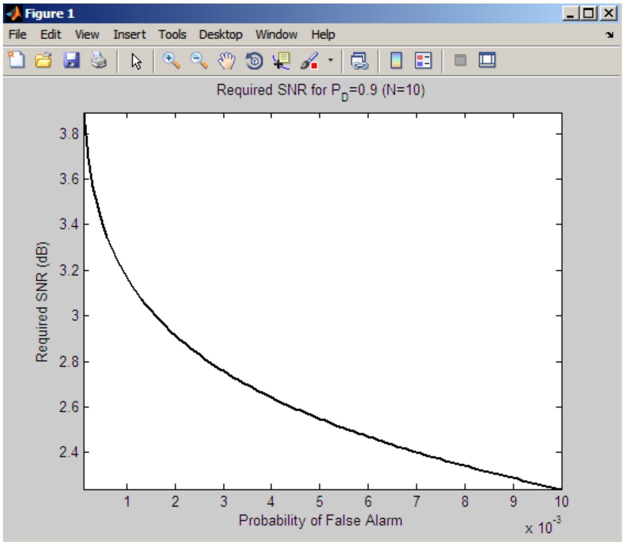
Compute the required single sample SNR for a detection probability of 0.9 as a function of the false-alarm probability.

```
Pfa=0.0001:0.0001:.01; % False-alarm probabilities
Pd=0.9; % probability of detection
SNR = zeros(1,length(Pfa)); % preallocate space
for j=1:length(Pfa)
    SNR(j) = albersheim(Pd,Pfa(j));
end
plot(Pfa,SNR,'k','linewidth',2);
axis tight;
xlabel('Probability of False Alarm');
ylabel('Required SNR (dB)');
title('Required SNR for P_D=0.9 (N=1)');
```



Compute the required SNR for 10 noncoherently integrated samples as a function of the false-alarm probability with the probability of detection equal to 0.9.

```
Pfa=0.0001:0.0001:.01; % False-alarm probabilities
Pd=0.9; % probability of detection
SNR = zeros(1,length(Pfa)); % preallocate space
for j=1:length(Pfa)
    SNR(j) = albersheim(Pd,Pfa(j),10);
end
plot(Pfa,SNR,'k','linewidth',2);
axis tight;
xlabel('Probability of False Alarm');
ylabel('Required SNR (dB)');
title('Required SNR for P_D=0.9 (N=10)');
```



References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing.* New York: McGraw-Hill, 2005, p. 329.

[2] Skolnik, M. *Introduction to Radar Systems*, 3rd Ed. New York: McGraw-Hill, 2001, p. 49.

See Also

`shnidman`

Purpose

Ambiguity function

Syntax

```
afmag = ambgfun(x,Fs,PRF)
[afmag,delay,doppler] = ambgfun(x,Fs,PRF)
[afmag,delay,doppler] = ambgfun(x,Fs,PRF,'Cut','2D')
[afmag,delay] = ambgfun(x,Fs,PRF,'Cut','Doppler')
[afmag,doppler] = ambgfun(x,Fs,PRF,'Cut','Delay')
ambgfun(x,Fs,PRF)
ambgfun(x,Fs,PRF,'Cut','2D')
ambgfun(x,Fs,PRF,'Cut','Delay')
ambgfun(x,Fs,PRF,'Cut','Doppler')
```

Description

`afmag = ambgfun(x,Fs,PRF)` returns the magnitude of the normalized ambiguity function for the vector `x`. The sampling of `x` occurs at `Fs` hertz with pulse repetition frequency, `PRF`. The sampling frequency `Fs` divided by the pulse repetition frequency `PRF` is the number of samples per pulse.

`[afmag,delay,doppler] = ambgfun(x,Fs,PRF)` or `[afmag,delay,doppler] = ambgfun(x,Fs,PRF,'Cut','2D')` returns the time delay vector, `delay`, and the Doppler frequency vector, `doppler`.

`[afmag,delay] = ambgfun(x,Fs,PRF,'Cut','Doppler')` returns the zero Doppler cut through the 2-D normalized ambiguity function magnitude.

`[afmag,doppler] = ambgfun(x,Fs,PRF,'Cut','Delay')` returns the zero delay cut through the 2-D normalized ambiguity function magnitude.

`ambgfun(x,Fs,PRF)` or `ambgfun(x,Fs,PRF,'Cut','2D')` with no output argument produces a contour plot of the ambiguity function.

`ambgfun(x,Fs,PRF,'Cut','Delay')` or `ambgfun(x,Fs,PRF,'Cut','Doppler')` with no output argument produces a line plot of the ambiguity function cut.

Input Arguments

x

Pulse waveform. `x` is a row or column vector.

Fs

Sampling frequency in hertz.

PRF

Pulse repetition frequency in hertz.

Output Arguments

afmag

Normalized ambiguity function magnitudes. `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 delay vector. `delay` is an N-by-1 vector of time delays. The time delay vector consists of `N = 2*length(x)-1` linearly spaced samples in the interval `(-length(x)/Fs, length(x)/Fs)`. The spacing between elements is the reciprocal of the sampling frequency.

doppler

Doppler frequency vector. `doppler` is an M-by-1 vector of Doppler frequencies. The Doppler frequency vector consists of linearly spaced samples in the frequency interval `[-Fs/2,Fs/2)`. The spacing between elements in the Doppler frequency vector is `Fs/2^nextpow2(2*length(x)-1)`.

Definitions

Normalized Ambiguity Function

The magnitude of the normalized ambiguity function is defined as:

$$|A(t,f_d)| = \frac{1}{E_x} \left| \int_{-\infty}^{\infty} x(u) e^{j2\pi f_d u} x^*(u-t)\, du \right|$$

where E_x is the norm of the signal, $x(t)$, t is the time delay, and f_d is a Doppler shift. The asterisk (*) denotes the complex conjugate.

The ambiguity function is a function of two variables that describes the effects of time delays and Doppler shifts on the output of a matched filter.

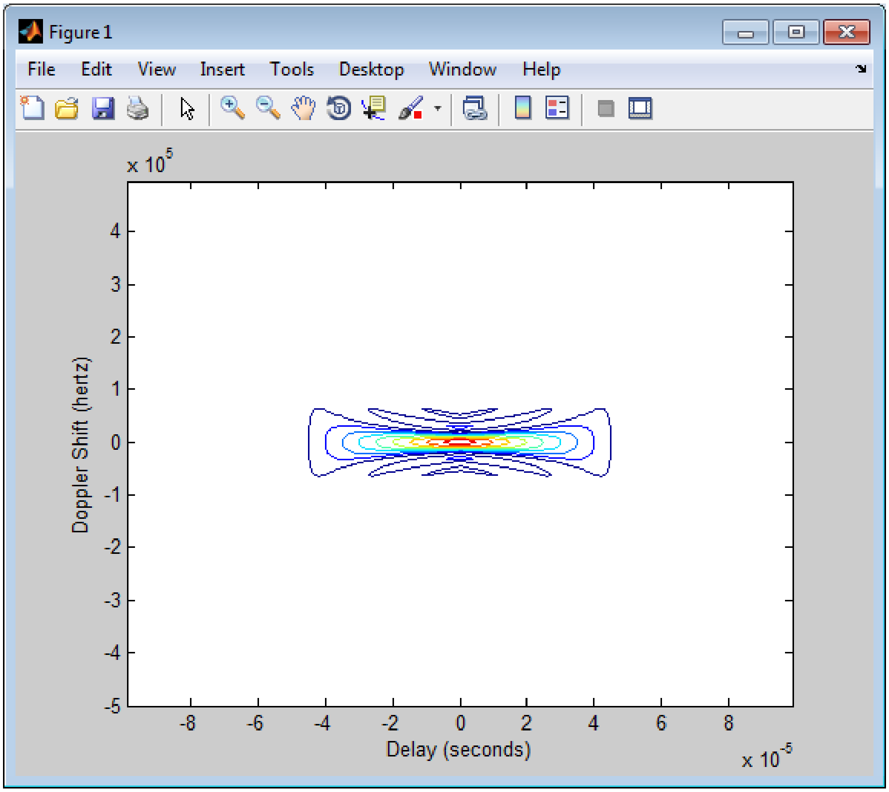
The magnitude of the ambiguity function at zero time delay and Doppler shift, $|A(0,0)|$, indicates the matched filter output when the received waveform exhibits the time delay and Doppler shift for which the matched filter is designed. Nonzero values of the time delay and Doppler shift variables indicate that the received waveform exhibits mismatches in time delay and Doppler shift from the matched filter.

The 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. In the normalized ambiguity function, the maximum value equals one.

Examples

Plot the ambiguity function magnitude of a rectangular pulse.

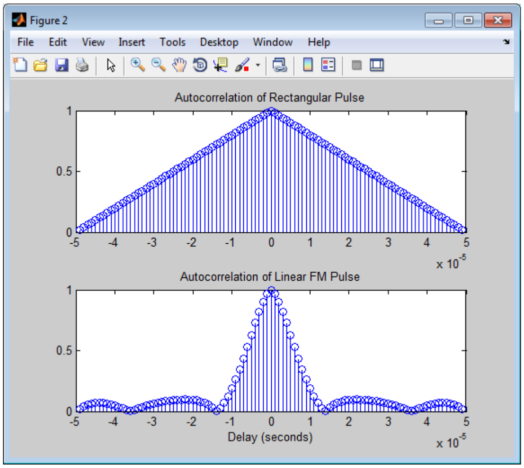
```
hrect = phased.RectangularWaveform;
% Default rectangular pulse waveform
x = step(hrect);
PRF = 2e4;
[afmag,delay,doppler] = ambgfun(x,hrect.SampleRate,PRF);
contour(delay,doppler,afmag);
xlabel('Delay (seconds)'); ylabel('Doppler Shift (hertz)');
```



Zero-Doppler cuts (autocorrelation sequences) for rectangular and linear FM pulses of the same 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 = step(hrect);
xfm = step(hfm);
[ambrect,delayrect] = ambgfun(xrect,hrect.SampleRate,...,
    hrect.PRF,'Cut','Doppler');
[ambfm,delayfm] = ambgfun(xfm,hfm.SampleRate,...,
```

```
    hfm.PRF,'Cut','Doppler');
figure;
subplot(211);
stem(delayrect,ambrect);
title('Autocorrelation of Rectangular Pulse');
subplot(212);
stem(delayfm,ambfm)
xlabel('Delay (seconds)');
title('Autocorrelation of Linear FM Pulse');
```



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

`phased.LinearFMWaveform` **|** `phased.MatchedFilter` **|** `phased.RectangularWaveform` **|** `phased.SteppedFMWaveform` **|**

aperture2gain

Purpose

Convert effective aperture to gain

Syntax

```
G = aperture2gain(A,lambda)
```

Description

`G = aperture2gain(A,lambda)` returns the antenna gain in decibels corresponding to an effective aperture of `A` square meters for an incident electromagnetic wave with wavelength `lambda` meters. `A` can be a scalar or vector. If `A` is a vector, `G` is a vector of the same size as `A`. The elements of `G` represent the gains for the corresponding elements of `A`. `lambda` must be a scalar.

Input Arguments

A

Antenna effective aperture in square meters. The effective aperture describes how much energy is captured from an incident electromagnetic plane wave. The argument describes the functional area of the antenna and is not equivalent to the actual physical area. For a fixed wavelength, the antenna gain is proportional to the effective aperture. `A` can be a scalar or vector. If `A` is a vector, each element of `A` is the effective aperture of a single antenna.

lambda

Wavelength of the incident electromagnetic wave. The wavelength of an electromagnetic wave is the ratio of the wave propagation speed to the frequency. For a fixed effective aperture, the antenna gain is inversely proportional to the square of the wavelength. `lambda` must be a scalar.

Output Arguments

G

Antenna gain in decibels. `G` is a scalar or a vector. If `G` is a vector, each element of `G` is the gain corresponding to effective aperture of the same element in `A`.

Definitions

Gain and Effective Aperture

The relationship between the gain, G, and effective aperture of an antenna, A_e is:

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

where λ is the wavelength of the incident electromagnetic wave. The gain expressed in decibels is:

$$10 \log_{10}(G)$$

Examples

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);
```

References

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

See Also

`gain2aperture`

az2broadside

Purpose

Convert azimuth angle to broadside angle

Syntax

```
BSang = az2broadside(az,el)
```

Description

`BSang = az2broadside(az,el)` returns the broadside angle `BSang` corresponding to the azimuth angle, `az`, and the elevation angle, `el`. All angles are expressed in degrees and in the local coordinate system. `az` and `el` can be either scalars or vectors. If both of them are vectors, their dimensions must match.

Definitions

Broadside Angle

The broadside angle ß corresponding to an azimuth angle *az* and an elevation angle *el* is:

$$\beta = \sin^{-1}(\sin(az)\cos(el))$$

where $-180 \leq az \leq 180$ and $-90 \leq el \leq 90$.

Examples

Broadside Angle for Scalar Inputs

Return the broadside angle corresponding to 45 degrees azimuth and 45 degrees elevation.

```
BSang = az2broadside(45,45);
```

Broadside Angles for Vector Inputs

Return broadside angles for 10 azimuth/elevation pairs. The variables `az`, `el`, and `BSang` are all 10-by-1 column vectors.

```
az = (75:5:120)';
el = (45:5:90)';
BSang = az2broadside(az,el);
```

See Also

`broadside2az` | `uv2azel` | `phitheta2azel`

azel2phitheta

Purpose

Convert angles from azimuth/elevation form to phi/theta form

Syntax

`PhiTheta = azel2phitheta(AzEl)`

Description

`PhiTheta = azel2phitheta(AzEl)` converts the azimuth/elevation angle pairs to their corresponding phi/theta angle pairs.

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

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`.

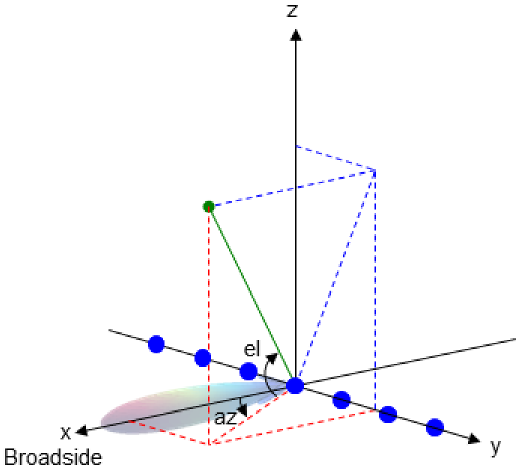
Definitions

Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive x-axis toward the positive y-axis, to the vector's orthogonal projection onto the xy plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the xy plane toward the positive z-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive x-axis.

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.

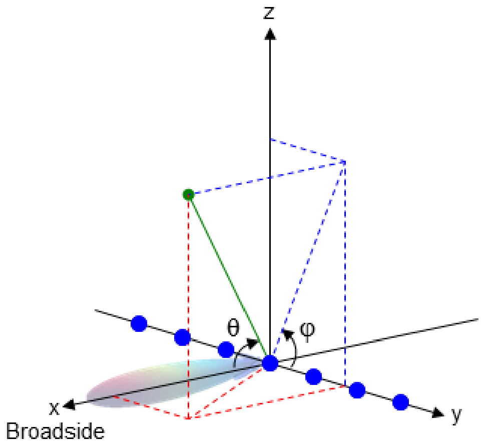
This figure illustrates the azimuth angle and elevation angle for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Phi Angle, Theta Angle

The φ angle is the angle from the positive *y*-axis toward the positive *z*-axis, to the vector's orthogonal projection onto the *yz* plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the *x*-axis toward the *yz* plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of Azimuth/Elevation Pair

Find the corresponding φ/θ representation for 30 degrees azimuth and 0 degrees elevation.

```
PhiTheta = azel2phitheta([30; 0]);
```

See Also

`phitheta2azel`

Concepts

- "Spherical Coordinates"

azel2phithetapat

Purpose

Convert radiation pattern from azimuth/elevation to phi/theta form

Syntax

```
pat_phitheta = azel2phithetapat(pat_azel,az,el)
pat_phitheta = azel2phithetapat(pat_azel,az,el,phi,theta)
[pat_phitheta,phi,theta] = azel2phithetapat( ___ )
```

Description

`pat_phitheta = azel2phithetapat(pat_azel,az,el)` expresses the antenna radiation pattern `pat_azel` in φ/θ angle coordinates instead of azimuth/elevation angle coordinates. `pat_azel` samples the pattern at azimuth angles in `az` and elevation angles in `el`. The `pat_phitheta` matrix covers φ values from 0 to 180 degrees and θ values from 0 to 360 degrees. `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 = azel2phithetapat(pat_azel,az,el,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, 180], and `theta` should cover the range [0, 360].

`[pat_phitheta,phi,theta] = azel2phithetapat( ___ )` 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.

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 –180 and 180.

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

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

`[0:180]` (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 180.

Data Types
double

Output Arguments

pat_phitheta - Antenna radiation pattern in phi/theta form

M-by-L matrix

Antenna radiation pattern in phi/theta form, returned as an M-by-L matrix. `pat_phitheta` samples the 3-D magnitude pattern in decibels, in terms of φ and θ angles. L is the length of the `phi` vector, and M is the length of the `theta` vector.

phi - Phi angles

vector of length L

Phi angles at which `pat_phitheta` samples the pattern, returned as a vector of length L. Angles are expressed in degrees.

theta - Theta angles

vector of length M

Theta angles at which `pat_phitheta` samples the pattern, returned as a vector of length M. Angles are expressed in degrees.

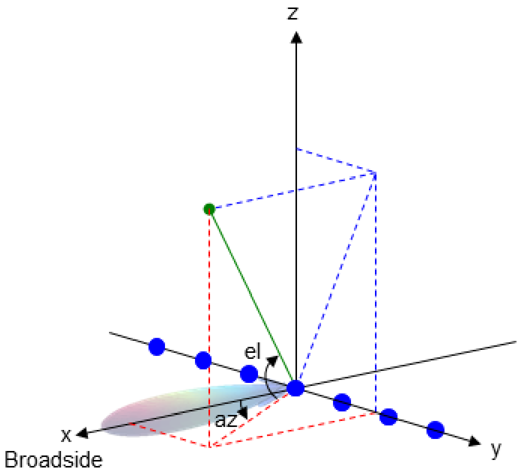
Definitions

Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive *x*-axis toward the positive *y*-axis, to the vector's orthogonal projection onto the *xy* plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the *xy* plane toward the positive *z*-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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.

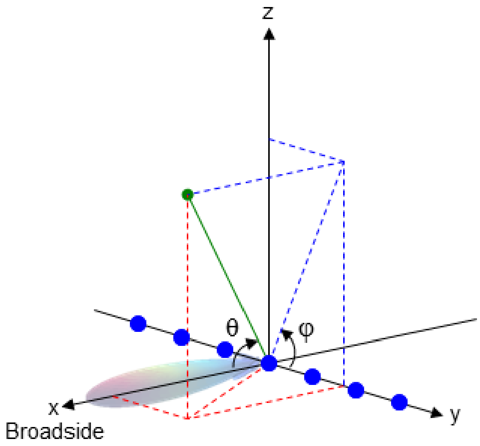
This figure illustrates the azimuth angle and elevation angle for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Phi Angle, Theta Angle

The φ angle is the angle from the positive *y*-axis toward the positive *z*-axis, to the vector's orthogonal projection onto the *yz* plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the *x*-axis toward the *yz* plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of Radiation Pattern

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 of Converted Radiation Pattern

Plot the result of converting 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;
```

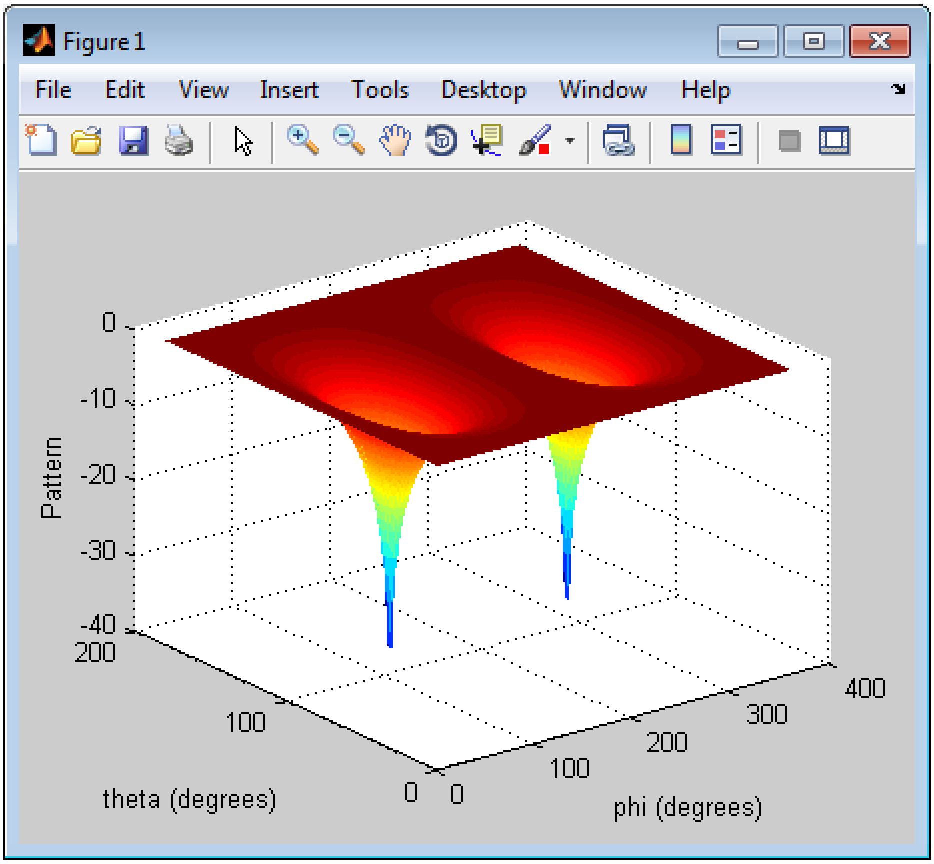
```
pat_azel = mag2db(repmat(cosd(el)',1,numel(az)));
```

Convert the pattern to φ/θ space. Store the φ and θ angles to use them for plotting.

```
[pat_phitheta,phi,theta] = azel2phithetapat(pat_azel,az,el);
```

Plot the result.

```
H = surf(phi,theta,pat_phitheta);
set(H,'LineStyle','none')
xlabel('phi (degrees)');
ylabel('theta (degrees)');
zlabel('Pattern');
```



Conversion of Radiation Pattern Using Specific Phi/Theta Values

Convert a radiation pattern to φ/θ form, with the φ and θ angles spaced 5 degrees apart.

Define the pattern in terms of azimuth and elevation.

```
az = -180:180;
el = -90:90;
```

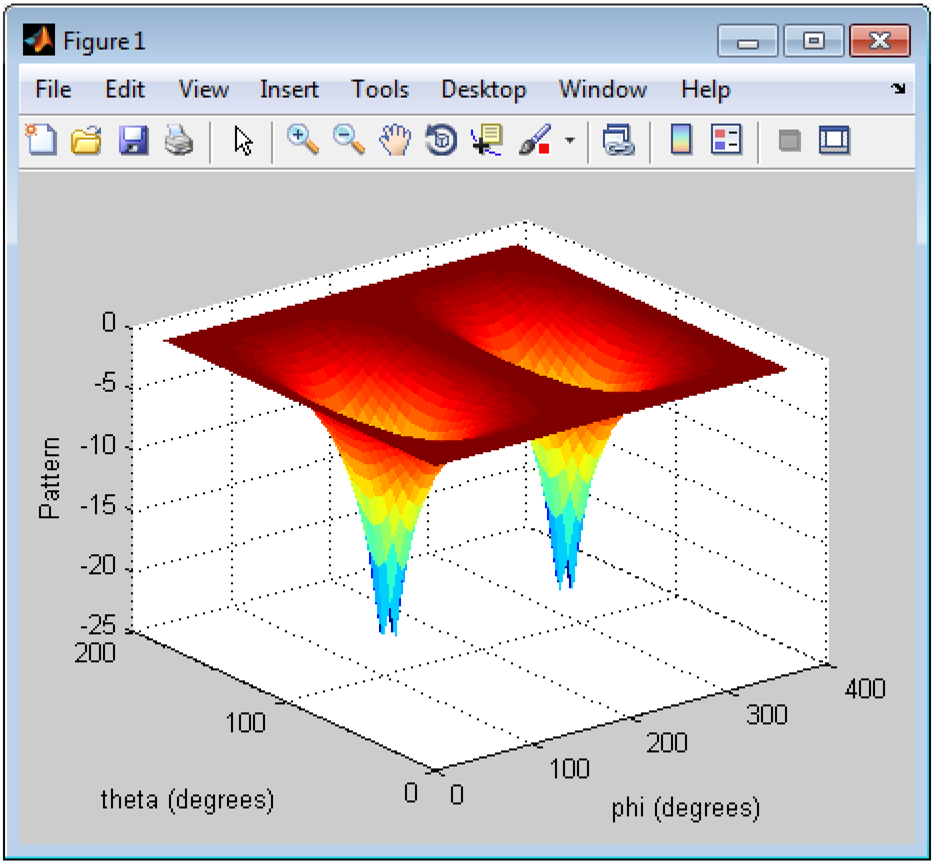
```
pat_azel = mag2db(repmat(cosd(el)',1,numel(az)));
```

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,pat_phitheta);
set(H,'LineStyle','none')
xlabel('phi (degrees)');
ylabel('theta (degrees)');
zlabel('Pattern');
```



See Also

`phased.CustomAntennaElement` | `phitheta2azel` | `azel2phitheta` | `phitheta2azelpat`

Concepts

- "Spherical Coordinates"

Purpose

Convert azimuth/elevation angles to u/v coordinates

Syntax

`UV = azel2uv(AzEl)`

Description

`UV = azel2uv(AzEl)` converts the azimuth/elevation angle pairs to their corresponding coordinates in *u*/*v* space.

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

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`.

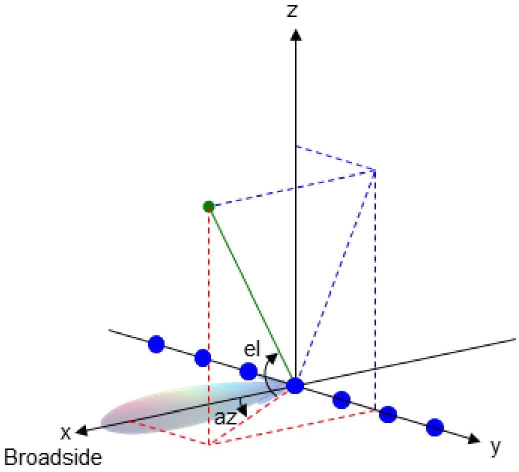
Definitions

Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive *x*-axis toward the positive *y*-axis, to the vector's orthogonal projection onto the *xy* plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the *xy* plane toward the positive *z*-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



U/V Space

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, as follows:

$$u = \sin(\theta)\cos(\varphi)$$

$$v = \sin(\theta)\sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

The values of *u* and *v* satisfy these inequalities:

$$-1 \le u \le 1$$

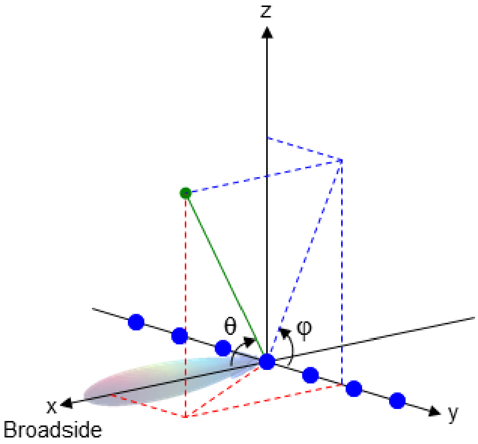
$$-1 \le v \le 1$$

$$u^2 + v^2 \le 1$$

Phi Angle, Theta Angle

The φ angle is the angle from the positive y-axis toward the positive z-axis, to the vector's orthogonal projection onto the yz plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the x-axis toward the yz plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of Azimuth/Elevation Pair

Find the corresponding u/v representation for 30 degrees azimuth and 0 degrees elevation.

```
UV = azel2uv([30; 0]);
```

See Also `uv2azel`

Concepts

- “Spherical Coordinates”

Purpose

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,v] = azel2uvpat( ___ )
```

Description

`pat_uv = azel2uvpat(pat_azel,az,el)` expresses the antenna radiation pattern `pat_azel` in u/v space coordinates instead of azimuth/elevation angle 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,v] = 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.

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` - *u* coordinates

vector of length L

u coordinates at which `pat_uv` samples the pattern, returned as a vector of length L.

`v` - *v* coordinates

vector of length M

v coordinates at which `pat_uv` samples the pattern, returned as a vector of length M.

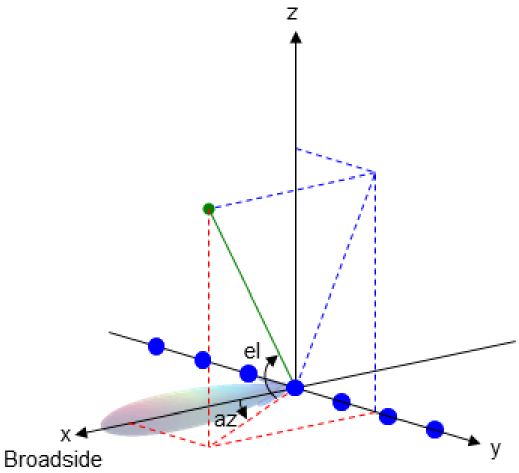
Definitions

Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive *x*-axis toward the positive *y*-axis, to the vector's orthogonal projection onto the *xy* plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the *xy* plane toward the positive *z*-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



U/V Space

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, as follows:

$$u = \sin(\theta)\cos(\varphi)$$

$$v = \sin(\theta)\sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

The values of u and v satisfy these inequalities:

$$-1 \leq u \leq 1$$

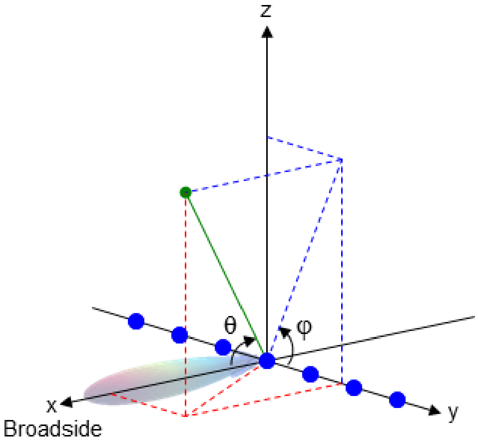
$$-1 \leq v \leq 1$$

$$u^2 + v^2 \leq 1$$

Phi Angle, Theta Angle

The φ angle is the angle from the positive *y*-axis toward the positive *z*-axis, to the vector's orthogonal projection onto the *yz* plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the *x*-axis toward the *yz* plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



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 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 of Converted Radiation Pattern

Plot the result of converting a radiation pattern to *u*/*v* form, with the *u* and *v* coordinates spaced by 0.01.

Define the pattern in terms of azimuth and elevation.

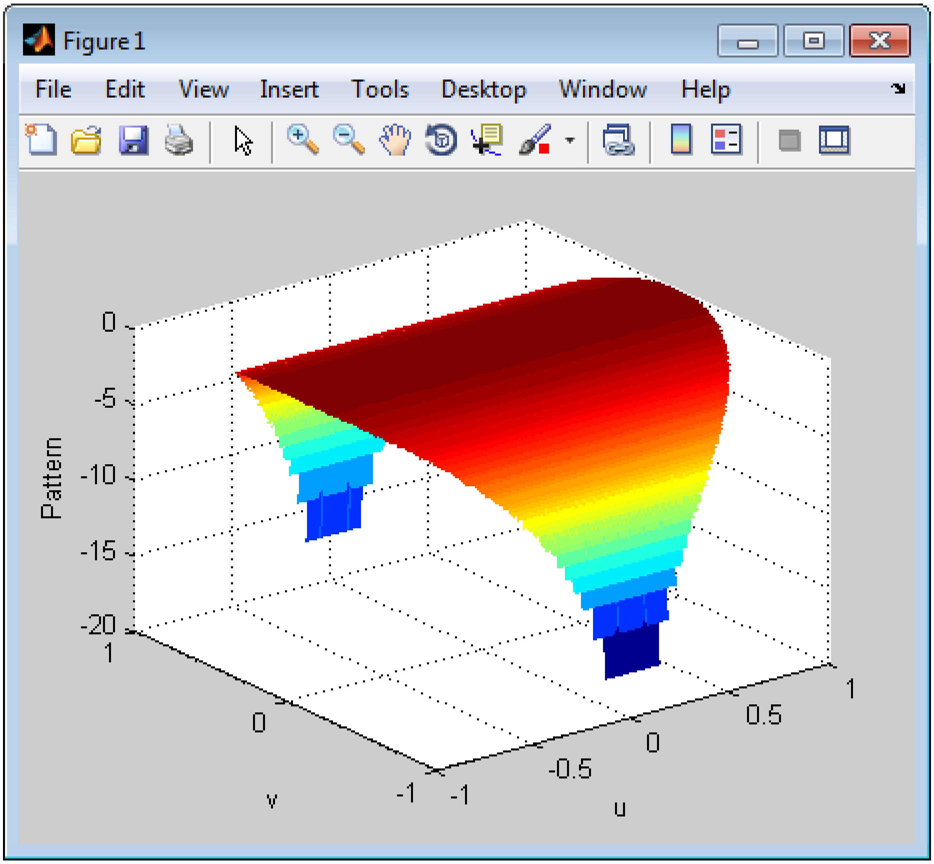
```
az = -90:90;
el = -90:90;
pat_azel = mag2db(repmat(cosd(el)',1,numel(az)));
```

Convert the pattern to *u*/*v* space. Store the *u* and *v* coordinates to use them for plotting.

```
[pat_uv,u,v] = azel2uvpat(pat_azel,az,el);
```

Plot the result.

```
H = surf(u,v,pat_uv);
set(H,'LineStyle','none')
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



Conversion of Radiation Pattern Using Specific U/V Values

Convert a radiation pattern to *u*/*v* form, with the *u* and *v* coordinates spaced by 0.05.

Define the pattern in terms of azimuth and elevation.

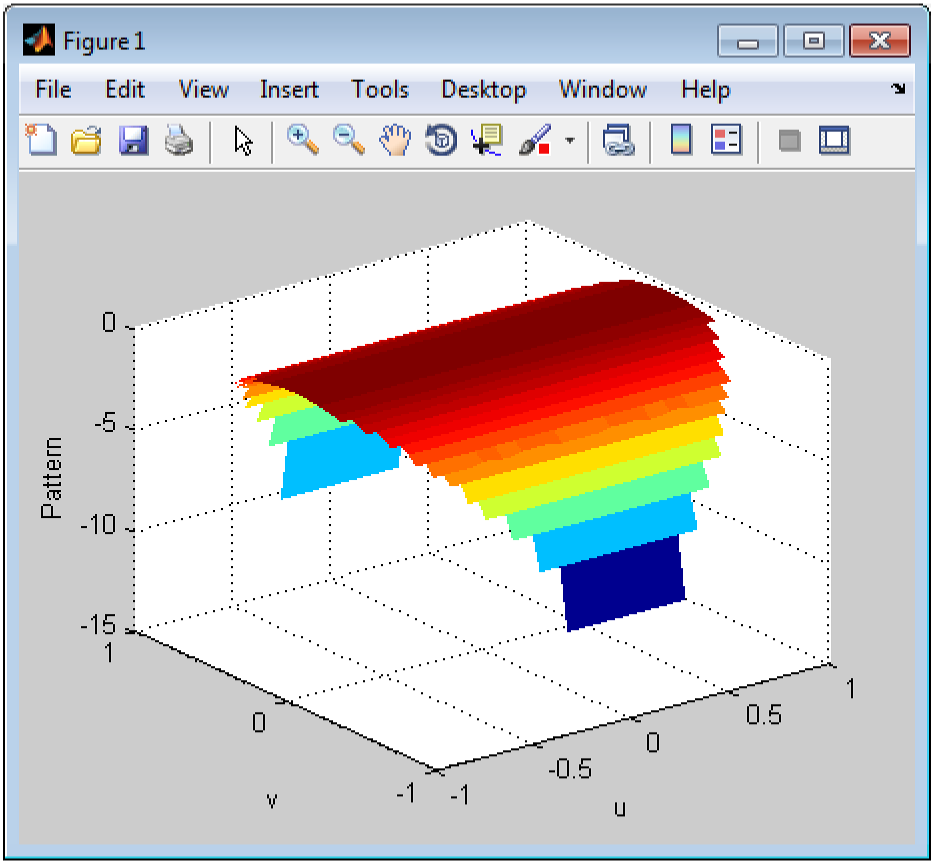
```
az = -90:90;
el = -90:90;
pat_azel = mag2db(repmat(cosd(el)',1,numel(az)));
```

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,pat_uv);
set(H,'LineStyle','none')
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



See Also `phased.CustomAntennaElement` | `azel2uv` | `uv2azel` | `uv2azelpat`

Concepts

- "Spherical Coordinates"

beat2range

Purpose

Convert beat frequency to range

Syntax

```
r = beat2range(fb,slope)
r = beat2range(fb,slope,c)
```

Description

`r = beat2range(fb,slope)` converts the beat frequency 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.

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
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
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

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`.

Definitions

Beat Frequency

For an upsweep or downsweep 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)`.

Examples

Range of Target in FMCW Radar System

Assume that the FMCW waveform sweeps a band of 3 MHz in 2 ms. The dechirped target return has a beat frequency of 1 kHz.

```
slope = 30e6/(2e-3);
fb = 1e3;
r = beat2range(fb,slope);
```

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.

See Also

`dechirp` | `range2beat` | `rdcoupling` `phased.FMCWWaveform` |

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

Purpose

Billingsley's intrinsic clutter motion (ICM) model

Syntax

```
P = billingsleyicm(fd,fc,wspeed)
P = billingsleyicm(fd,fc,wspeed,c)
```

Description

`P = billingsleyicm(fd,fc,wspeed)` calculates the clutter Doppler spectrum shape, `P`, due to intrinsic clutter motion (ICM) at Doppler frequencies specified in `fd`. ICM arises when wind blows on vegetation or other clutter sources. This function uses Billingsley's model in the calculation. `fc` is the operating frequency of the system. `wspeed` is the wind speed.

`P = billingsleyicm(fd,fc,wspeed,c)` specifies the propagation speed `c` in meters per second.

Input Arguments

fd

Doppler frequencies in hertz. This value can be a scalar or a vector.

fc

Operating frequency of the system in hertz

wspeed

Wind speed in meters per second

c

Propagation speed in meters per second

Default: Speed of light

Output Arguments

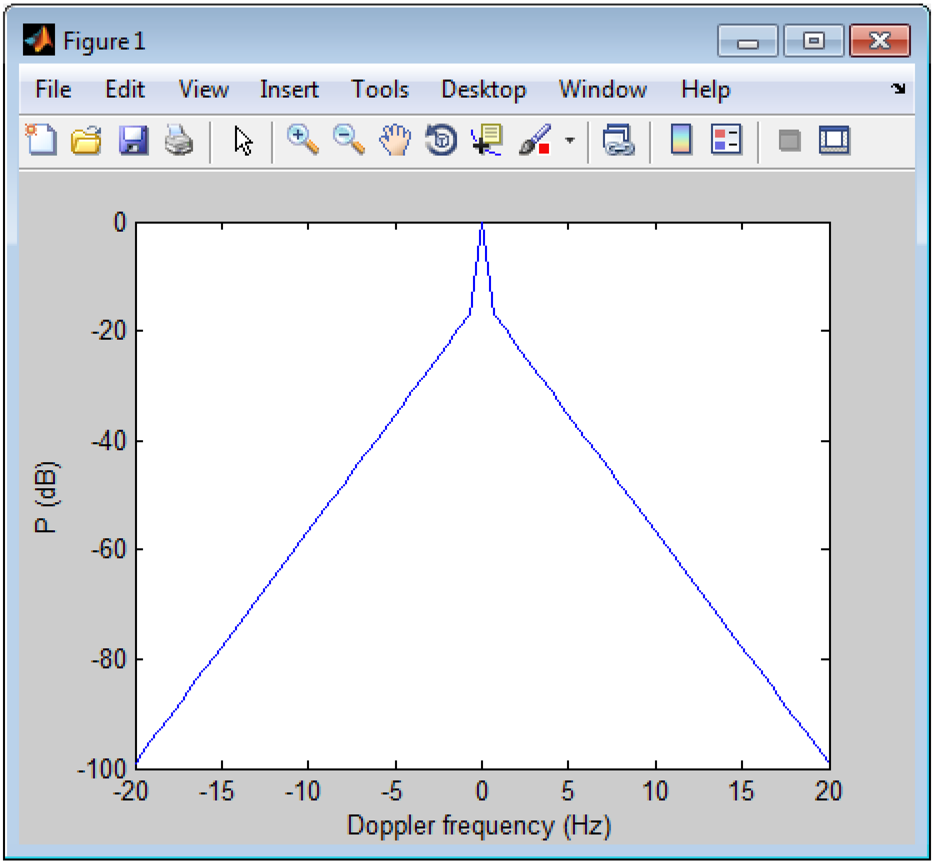
P

Shape of the clutter Doppler spectrum due to intrinsic clutter motion. The vector size of `P` is the same as that of `fd`.

Examples

Calculate and plot the Doppler spectrum shape predicted by Billingsley's ICM model. Assume the PRF is 2 kHz, the operating frequency is 1 GHz, and the wind speed is 5 m/s.

```
v = -3:0.1:3; fc = 1e9; wspeed = 5; c = 3e8;
fd = 2*v/(c/fc);
p = billingsleyicm(fd,fc,wspeed);
plot(fd,pow2db(p));
xlabel('Doppler frequency (Hz)'), ylabel('P (dB)');
```



References

[1] Billingsley, J. *Low Angle Radar Clutter*. Norwich, NY: William Andrew Publishing, 2002.

[2] Long, Maurice W. *Radar Reflectivity of Land and Sea*, 3rd Ed. Boston: Artech House, 2001.

broadside2az

Purpose

Convert broadside angle to azimuth angle

Syntax

```
az = broadside2az(BSang,el)
```

Description

`az = broadside2az(BSang,el)` returns the azimuth angle, `az`, corresponding to the broadside angle `BSang` and the elevation angle, `el`. All angles are in degrees and in the local coordinate system. `BSang` and `el` can be either scalars or vectors. If both of them are vectors, their dimensions must match.

Definitions

Azimuth Angle

The azimuth angle *az* corresponding to a broadside angle ß and elevation angle *el* is:

$$az = \sin^{-1}(\sin(\beta)\sec(el))$$

where $-90 \le el \le 90$, $-90 \le \beta \le 90$, and $-180 \le az \le 180$.

Together the broadside and elevation angles must satisfy the following inequality:

$$|\beta| + |el| \le 90$$

Examples

Azimuth Angle for Scalar Inputs

Return the azimuth angle corresponding to a broadside angle of 45 degrees and an elevation angle of 20 degrees.

```
az = broadside2az(45,20);
```

Azimuth Angles for Vector Inputs

Return azimuth angles for 10 pairs of broadside angle and elevation angle. The variables `BSang`, `el`, and `az` are all 10-by-1 column vectors.

```
BSang = (45:5:90)';
el = (45:-5:0)';
az = broadside2az(BSang,el);
```

See Also

`az2broadside` | `azel2uv` | `azel2phitheta`

Purpose

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.

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
double
Complex Number Support: Yes

xref - Reference signal

M-by-1 vector

Reference signal, specified as an M-by-1 vector.

Data Types
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`.

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);
```

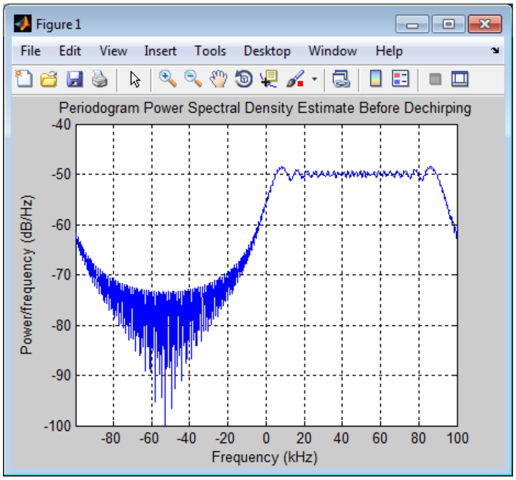
Dechirp a delayed copy of the signal.

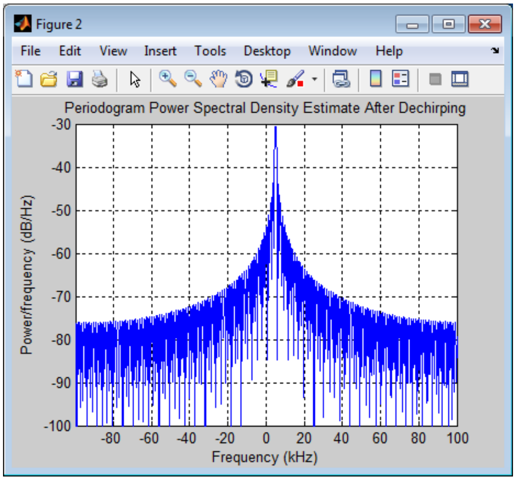
```
x = [zeros(10,1); xref(1:end-10)];
y = dechirp(x,xref);
```

Plot the spectrum before and after dechirping.

```
figure;
psd(spectrum.periodogram,x,'NFFT',1024,'Fs',Fs,'CenterDC',true);
title('Periodogram Power Spectral Density Estimate Before Dechirping')

figure;
psd(spectrum.periodogram,y,'NFFT',1024,'Fs',Fs,'CenterDC',true);
ylim([-100 -30]);
title('Periodogram Power Spectral Density Estimate After Dechirping')
```





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 negates the Doppler shift embedded in `x`, because of the order of `xref` and `x`.

The mixing order affects the sign of the imaginary part of `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.

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.

See Also

`beat2range` | `phased.RangeDopplerResponse` |

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

Purpose

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 input `data` by `DELAY` samples. Negative values of `DELAY` advance `data`, while positive values delay `data`. Noninteger values of `DELAY` represent fractional delays or advances. In this case, the function interpolates. How the `delayseq` function operates on the columns of `data` depends on the dimensions of `data` and `DELAY`:

- If `DELAY` is a scalar, the function applies that shift to each column of `data`.
- If `DELAY` is a vector whose length equals the number of columns of `data`, the function shifts each column by the corresponding vector entry.
- If `DELAY` is a vector and `data` has one column, the function shifts `data` by each entry in `DELAY` independently. The number of columns in `shifted_data` is the vector length of `DELAY`. 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 `DELAY` is not divisible by the reciprocal of the sampling frequency, `delayseq` interpolates to implement a fractional delay or advance of `data`.

Input Arguments

data

Vector or matrix of real or complex data.

DELAY

Amount by which to delay or advance the input. If you specify the optional `Fs` argument, `DELAY` is in seconds; otherwise, `DELAY` is in samples.

Fs

Sampling frequency of the data in hertz. If you specify this argument, the function assumes `DELAY` is in seconds.

Default: `1`

Output Arguments

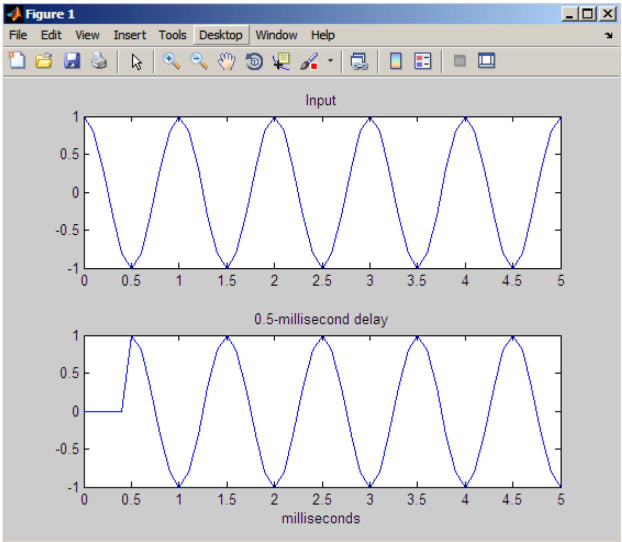
shifted_data

Result of delaying or advancing the data. `shifted_data` has the same number of rows as `data`, with appropriate truncations or zero padding.

Examples

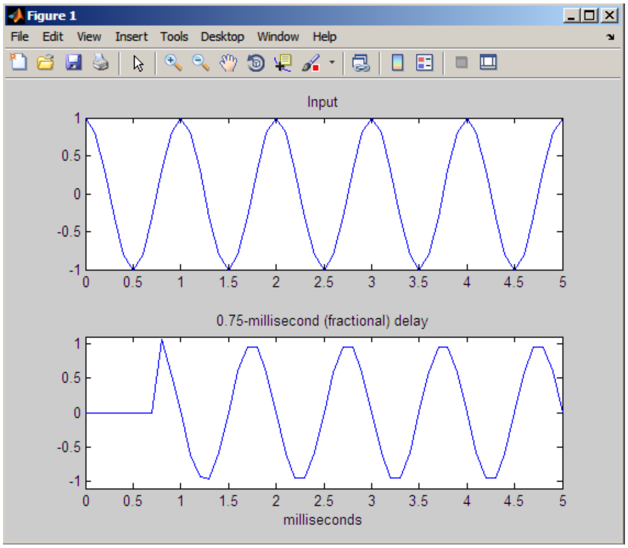
Implement integer delay of input sequence in seconds.

```
Fs = 1e4;
t = 0:1/Fs:0.005;
data = cos(2*pi*1000*t)'; % data is a column vector
% Delay input by 0.5 milliseconds (5 samples)
shifted_data = delayseq(data,0.0005,Fs);
subplot(211);
plot(t.*1000,data); title('Input');
subplot(212);
plot(t.*1000,shifted_data); title('0.5-millisecond delay');
xlabel('milliseconds');
```



Implement fractional delay of input sequence in seconds.

```
Fs = 1e4;
t = 0:1/Fs:0.005;
data = cos(2*pi*1000*t)'; % data is a column vector
% Delay input by 0.75 milliseconds (7.5 samples)
shifted_data = delayseq(data,0.00075,Fs);
figure;
subplot(211);
plot(t.*1000,data); title('Input');
subplot(212);
plot(t.*1000,shifted_data);
title('0.75-millisecond (fractional) delay');
axis([0 5 -1.1 1.1]); xlabel('milliseconds');
```



Note that the values of the shifted sequence differ from the input because of the interpolation resulting from the fractional delay.

See Also

`phased.TimeDelayBeamformer` |

depressionang

Purpose

Depression angle of surface target

Syntax

```
depAng = depressionang(H,R)
depAng = depressionang(H,R,MODEL)
depAng = depressionang(H,R,MODEL,Re)
```

Description

`depAng = depressionang(H,R)` returns the depression angle from the horizontal at an altitude of `H` meters to surface targets. The sensor is `H` meters above the surface. `R` is the range from the sensor to the surface targets. The computation assumes a curved earth model with an effective earth radius of approximately 4/3 times the actual earth radius.

`depAng = depressionang(H,R,MODEL)` specifies the earth model used to compute the depression angle. `MODEL` is either `'Flat'` or `'Curved'`.

`depAng = depressionang(H,R,MODEL,Re)` specifies the effective earth radius. Effective earth radius applies to a curved earth model. When `MODEL` is `'Flat'`, the function ignores `Re`.

Input Arguments

H

Height of the sensor above the surface, in meters. This argument can be a scalar or a vector. If both `H` and `R` are nonscalar, they must have the same dimensions.

R

Distance in meters from the sensor to the surface target. This argument can be a scalar or a vector. If both `H` and `R` are nonscalar, they must have the same dimensions. `R` must be between `H` and the horizon range determined by `H`.

MODEL

Earth model, as one of `| 'Curved' | 'Flat' |`.

Default: `'Curved'`

Re

Effective earth radius in meters. This argument requires a positive scalar value.

> **Default:** `effearthradius`, which is approximately 4/3 times the actual earth radius

Output Arguments

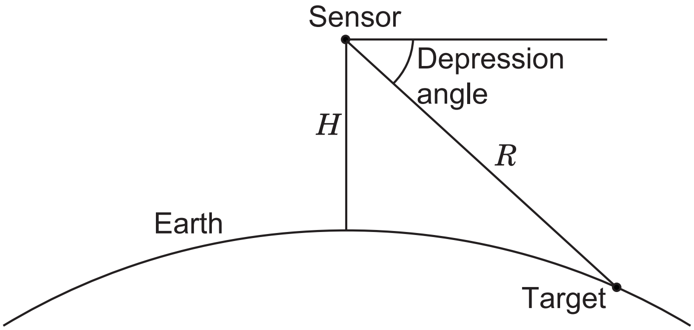
depAng

Depression angle, in degrees, from the horizontal at the sensor altitude toward surface targets `R` meters from the sensor. The dimensions of `depAng` are the larger of `size(H)` and `size(R)`.

Definitions

Depression Angle

The depression angle is the angle between a horizontal line containing the sensor and the line from the sensor to a surface target.



For the curved earth model with an effective earth radius of R_e, the depression angle is:

$$\sin^{-1}\left(\frac{H^2 + 2HR_e + R^2}{2R(H + R_e)}\right)$$

For the flat earth model, the depression angle is:

$$\sin^{-1}\left(\frac{H}{R}\right)$$

Examples

Calculate the depression angle for a ground clutter patch that is 1000 m away from the sensor. The sensor is located on a platform that is 300 m above the ground.

```
depang = depressionang(300,1000);
```

References

[1] Long, Maurice W. *Radar Reflectivity of Land and Sea*, 3rd Ed. Boston: Artech House, 2001.

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

See Also

`grazingang` | `horizonrange`

dop2speed

Purpose

Convert Doppler shift to speed

Syntax

`radvel = dop2speed(Doppler_shift,wavelength)`

Description

`radvel = dop2speed(Doppler_shift,wavelength)` returns the radial velocity in meters per second. This value corresponds to the one-way Doppler shift, `Doppler_shift`, for the wavelength `wavelength` in meters.

Definitions

The following equation defines the speed of a source relative to a receiver based on 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.

Examples

Calculate the speed of an automobile for continuous-wave radar based on the Doppler shift.

```
f0=24.15e9; % 24.15 GHz carrier
lambda=physconst('LightSpeed')/f0; % wavelength
% Assume Doppler shift of 2880 Hz
radvel = dop2speed(2880,lambda);
% Roughly 35.75 meters per second (80 miles/hour)
```

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

Purpose

Doppler steering vector

Syntax

```
DSTV = dopsteeringvec(dopplerfreq,numpulses)
DSTV = dopsteeringvec(dopplerfreq,numpulses,PRF)
```

Description

`DSTV = dopsteeringvec(dopplerfreq,numpulses)` returns the N-by-1 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`.

Input Arguments

dopplerfreq

The Doppler frequency in hertz. The normalized Doppler frequency is the Doppler frequency divided by the pulse repetition frequency.

numpulses

The number of pulses. The time-domain Doppler steering vector consists of `numpulses` samples taken at intervals of `1/PRF` (slow-time samples).

PRF

Pulse repetition frequency in hertz. 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.

Output Arguments

DSTV

Temporal (time-domain) Doppler steering vector. `DSTV` is an N-by-1 column vector where N is the number of pulses, `numpulses`.

Definitions

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 pulse), 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.

Examples

Calculate the steering vector corresponding to a Doppler frequency of 200 Hz, assuming there are 10 pulses and the PRF is 1 kHz.

```
dstv = dopsteeringvec(200,10,1000);
```

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`

effearthradius

Purpose

Effective earth radius

Syntax

```
Re = effearthradius
Re = effearthradius(RGradient)
```

Description

`Re = effearthradius` returns the effective radius of spherical earth in meters. The calculation uses a refractivity gradient of `-39e-9`. As a result, `Re` is approximately 4/3 of the actual earth radius.

`Re = effearthradius(RGradient)` specifies the refractivity gradient.

Input Arguments

RGradient

Refractivity gradient in units of 1/meter. This value must be a nonpositive scalar.

Default: `-39e-9`

Output Arguments

Re

Effective earth radius in meters.

Definitions

Effective Earth Radius

The *effective earth radius* is a scaling of the actual earth radius. The scale factor is:

$$\frac{1}{1 + r \cdot \mathtt{RGradient}}$$

where r is the actual earth radius in meters and `RGradient` is the refractivity gradient. The refractivity gradient, which depends on the altitude, is the rate of change of refraction index with altitude. The *refraction index* for a given altitude is the ratio between the free-space propagation speed and the propagation speed in the air band at that altitude.

The most commonly used scale factor is 4/3. This value corresponds to a refractivity gradient of $-39 \times 10^{-9}\ \mathrm{m}^{-1}$.

References

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

See Also

`depressionang` | `horizonrange`

Purpose

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 0, indicating no path loss.

Input Arguments

R

Propagation distance in meters

lambda

Wavelength in meters. The wavelength in meters is the speed of propagation divided by the frequency in hertz.

Output Arguments

L

Path loss in decibels. `L` is a nonnegative number. The minimum value of `L` is 0, indicating no path loss.

Definitions

Free Space Path Loss

The free space path loss, L, in decibels is:

$$L = 20\log_{10}(\frac{4\pi R}{\lambda})$$

Examples

Calculate free space path loss in decibels incurred by a 10 gigahertz wave over a distance of 10 kilometers.

```
lambda = physconst('LightSpeed')/10e9;
R = 10e3;
L = fspl(R,lambda);
```

References

[1] Proakis, J. *Digital Communications*. New York: McGraw-Hill, 2001.

See Also `phased.FreeSpace` |

gain2aperture

Purpose

Convert gain to effective aperture

Syntax

`A = gain2aperture(G,lambda)`

Description

`A = gain2aperture(G,lambda)` returns the effective aperture in square meters corresponding to a gain of `G` decibels for an incident electromagnetic wave with wavelength `lambda` meters. `G` can be a scalar or vector. If `G` is a vector, `A` is a vector of the same size as `G`. The elements of `A` represent the effective apertures for the corresponding elements of `G`. `lambda` must be a scalar.

Input Arguments

G

Antenna gain in decibels. `G` is a scalar or a vector. If `G` is a vector, each element of `G` is the gain in decibels of a single antenna.

lambda

Wavelength of the incident electromagnetic wave. The wavelength of an electromagnetic wave is the ratio of the wave propagation speed to the frequency. For a fixed effective aperture, the antenna gain is inversely proportional to the square of the wavelength. `lambda` must be a scalar.

Output Arguments

A

Antenna effective aperture in square meters. The effective aperture describes how much energy is captured from an incident electromagnetic plane wave. The argument describes the functional area of the antenna and is not equivalent to the actual physical area. For a fixed wavelength, the antenna gain is proportional to the effective aperture. `A` can be a scalar or vector. If `A` is a vector, each element of `A` is the effective aperture of the corresponding gain in `G`.

Definitions

Gain and Effective Aperture

The relationship between the gain, G, in decibels of an antenna and the antenna's effective aperture is:

$$A_e = 10^{G/10} \frac{\lambda^2}{4\pi}$$

where λ is the wavelength of the incident electromagnetic wave.

Examples

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);
```

References

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

See Also

`aperture2gain`

global2localcoord

Purpose

Convert global to local coordinates

Syntax

```
lclCoord = global2localcoord(gCoord, OPTION)
gCoord = global2localcoord( ___ ,localOrigin)
gCoord = global2localcoord( ___ ,localAxes)
```

Description

`lclCoord = global2localcoord(gCoord, OPTION)` returns the local coordinate `lclCoord` corresponding to the global coordinate `gCoord`. `OPTION` determines the type of global-to-local coordinate transformation.

`gCoord = global2localcoord( ___ ,localOrigin)` specifies the origin of the local coordinate system.

`gCoord = global2localcoord( ___ ,localAxes)` specifies the axes of the local coordinate system.

Input Arguments

gCoord

Global coordinates in rectangular or spherical coordinate form. `gCoord` is a 3-by-1 vector or 3-by-N matrix. Each column represents a global coordinate.

If the coordinates are in rectangular form, the column represents (X,Y,Z) in meters.

If the coordinates are in spherical form, the column represents (az,el,r). az is the azimuth angle in degrees, el is the elevation angle in degrees, and r is the radius in meters.

The origin of the global coordinate system is at [0; 0; 0]. That system's axes are the standard unit basis vectors in three-dimensional space, [1; 0; 0], [0; 1; 0], and [0; 0; 1].

OPTION

Type of coordinate transformation. Valid strings are in the next table.

OPTION	Transformation
`'rr'`	Global rectangular to local rectangular
`'rs'`	Global rectangular to local spherical
`'sr'`	Global spherical to local rectangular
`'ss'`	Global spherical to local spherical

localOrigin

Origin of local coordinate system. `localOrigin` is a 3-by-1 column vector containing the rectangular coordinate of the local coordinate system origin with respect to the global coordinate system.

Default: `[0; 0; 0]`

localAxes

Axes of local coordinate system. `localAxes` is a 3-by-3 matrix with the columns specifying the local X, Y, and Z axes in rectangular form with respect to the global coordinate system.

Default: `[1 0 0;0 1 0;0 0 1]`

Output Arguments

lclCoord

Local coordinates in rectangular or spherical coordinate form.

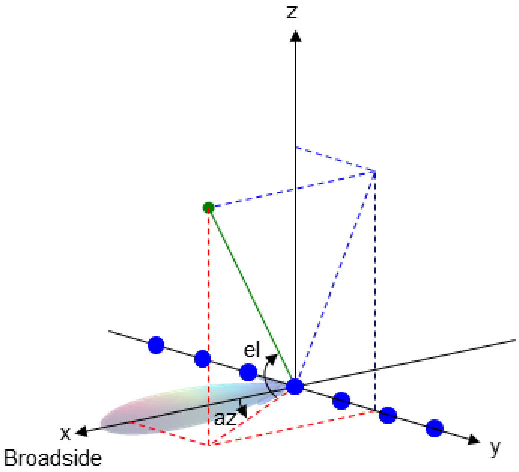
Definitions

Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive x-axis toward the positive y-axis, to the vector's orthogonal projection onto the xy plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the xy plane toward the positive z-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Convert between global and local coordinates in rectangular form.

```
lclCoord = global2localcoord([0; 1; 0], ...
'rr',[1; 1; 1]);
% Local origin is at [1; 1; 1]
% lclCoord = [0; 1; 0]-[1; 1; 1];
```

Convert global spherical coordinate to local rectangular coordinate.

```
lclCoord = global2localcoord([45; 45; 50],'sr',[50; 50; 50]);
% 45 degree azimuth, 45 degree elevation, 50 meter radius
```

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.

See Also

`local2globalcoord` | `uv2azel` | `phitheta2azel` | `azel2uv` | `azel2phitheta`

Concepts

- "Global and Local Coordinate Systems"

grazingang

Purpose

Grazing angle of surface target

Syntax

```
grazAng = grazingang(H,R)
grazAng = grazingang(H,R,MODEL)
grazAng = grazingang(H,R,MODEL,Re)
```

Description

`grazAng = grazingang(H,R)` returns the grazing angle for a sensor `H` meters above the surface, to surface targets `R` meters away. The computation assumes a curved earth model with an effective earth radius of approximately 4/3 times the actual earth radius.

`grazAng = grazingang(H,R,MODEL)` specifies the earth model used to compute the grazing angle. `MODEL` is either `'Flat'` or `'Curved'`.

`grazAng = grazingang(H,R,MODEL,Re)` specifies the effective earth radius. Effective earth radius applies to a curved earth model. When `MODEL` is `'Flat'`, the function ignores `Re`.

Input Arguments

H

Height of the sensor above the surface, in meters. This argument can be a scalar or a vector. If both `H` and `R` are nonscalar, they must have the same dimensions.

R

Distance in meters from the sensor to the surface target. This argument can be a scalar or a vector. If both `H` and `R` are nonscalar, they must have the same dimensions. `R` must be between `H` and the horizon range determined by `H`.

MODEL

Earth model, as one of `| 'Curved' | 'Flat' |`.

Default: `'Curved'`

Re

Effective earth radius in meters. This argument requires a positive scalar value.

> **Default:** `effearthradius`, which is approximately 4/3 times the actual earth radius

Output Arguments

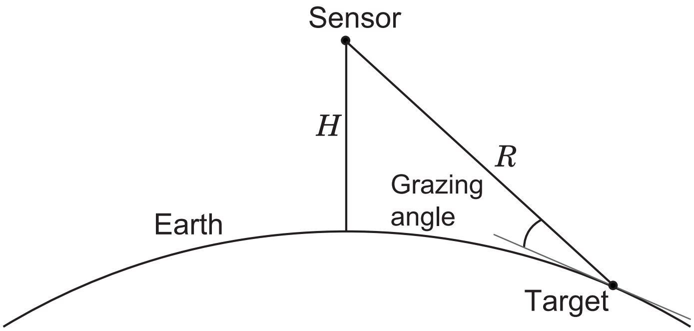
grazAng

Grazing angle, in degrees. The size of `grazAng` is the larger of `size(H)` and `size(R)`.

Definitions

Grazing Angle

The grazing angle is the angle between a line from the sensor to a surface target, and a tangent to the earth at the site of that target.



For the curved earth model with an effective earth radius of R_e, the grazing angle is:

$$\sin^{-1}\left(\frac{H^2 + 2HR_e - R^2}{2RR_e}\right)$$

For the flat earth model, the grazing angle is:

$$\sin^{-1}\left(\frac{H}{R}\right)$$

Examples

Determine the grazing angle of a ground target located 1000 m away from the sensor. The sensor is mounted on a platform that is 300 m above the ground.

```
grazAng = grazingang(300,1000);
```

References

[1] Long, Maurice W. *Radar Reflectivity of Land and Sea*, 3rd Ed. Boston: Artech House, 2001.

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

See Also

`depressionang` | `horizonrange`

Purpose

Horizon range

Syntax

```
Rh = horizonrange(H)
Rh = horizonrange(H,Re)
```

Description

`Rh = horizonrange(H)` returns the horizon range of a radar system `H` meters above the surface. The computation uses an effective earth radius of approximately 4/3 times the actual earth radius.

`Rh = horizonrange(H,Re)` specifies the effective earth radius.

Input Arguments

H

Height of radar system above surface, in meters. This argument can be a scalar or a vector.

Re

Effective earth radius in meters. This argument must be a positive scalar.

> **Default:** `effearthradius`, which is approximately 4/3 times the actual earth radius

Output Arguments

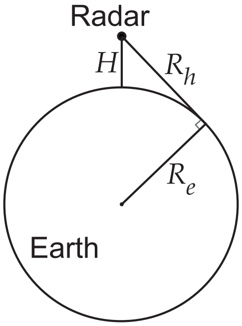
Rh

Horizon range in meters of radar system at altitude `H`.

Definitions

Horizon Range

The *horizon range* of a radar system is the distance from the radar system to the earth along a tangent. Beyond the horizon range, the radar system detects no return from the surface through a direct path.



The value of the horizon range is:

$$\sqrt{2R_eH + H^2}$$

where R_e is the effective earth radius and H is the altitude of the radar system.

Examples

Determine the horizon range of an antenna that is 30 m high.

```
Rh = horizonrange(30);
```

References

[1] Long, Maurice W. *Radar Reflectivity of Land and Sea*, 3rd Ed. Boston: Artech House, 2001.

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

See Also

`depressionang` | `effearthradius` | `grazingang`

Purpose

Convert local to global coordinates

Syntax

```
gCoord = local2globalcoord(lclCoord,OPTION)
gCoord = local2globalcoord(___,localOrigin)
gCoord = local2globalcoord(___,localAxes)
```

Description

`gCoord = local2globalcoord(lclCoord,OPTION)` returns the global coordinate `gCoord` corresponding to the local coordinate `lclCoord`. `OPTION` determines the type of local-to-global coordinate transformation.

`gCoord = local2globalcoord(___,localOrigin)` specifies the origin of the local coordinate system.

`gCoord = local2globalcoord(___,localAxes)` specifies the axes of the local coordinate system.

Input Arguments

lclCoord

Local coordinates in rectangular or spherical coordinate form. `lclCoord` is a 3-by-1 vector or 3-by-N matrix. Each column represents a local coordinate.

If the coordinates are in rectangular form, the column represents (X,Y,Z) in meters.

If the coordinates are in spherical form, the column represents (az,el,r). az is the azimuth angle in degrees, el is the elevation angle in degrees, and r is the radius in meters.

OPTION

Type of coordinate transformation. Valid strings are in the next table.

OPTION	Transformation
`'rr'`	Local rectangular to global rectangular
`'rs'`	Local rectangular to global spherical
`'sr'`	Local spherical to global rectangular
`'ss'`	Local spherical to global spherical

localOrigin

Origin of local coordinate system. `localOrigin` is a 3-by-1 column vector containing the rectangular coordinate of the local coordinate system origin with respect to the global coordinate system.

Default: `[0; 0; 0]`

localAxes

Axes of local coordinate system. `localAxes` is a 3-by-3 matrix with the columns specifying the local X, Y, and Z axes in rectangular form with respect to the global coordinate system.

Default: `[1 0 0;0 1 0;0 0 1]`

Output Arguments

gCoord

Global coordinates in rectangular or spherical coordinate form. The origin of the global coordinate system is at [0; 0; 0]. That system's axes are the standard unit basis vectors in three-dimensional space, [1; 0; 0], [0; 1; 0], and [0; 0; 1].

Definitions

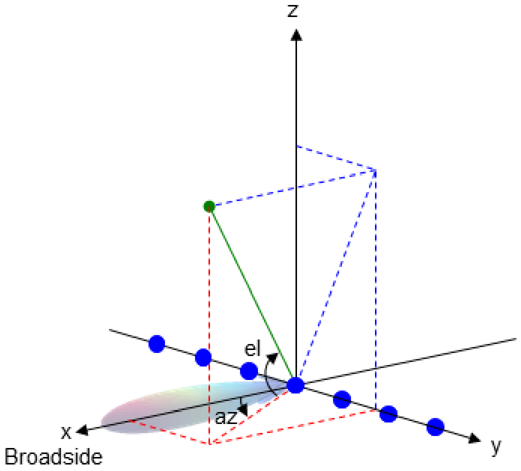
Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive *x*-axis toward the positive *y*-axis, to the vector's orthogonal projection onto the *xy* plane.

The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the *xy* plane toward the positive *z*-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Convert between local and global coordinate in rectangular form.

```
gCoord = local2globalcoord([0; 1; 0], ...
'rr',[1; 1; 1]);
```

```
% Local origin is at [1; 1; 1]
% gCoord = [1 1 1]+[0 1 0];
```

Convert local spherical coordinate to global rectangular coordinate.

```
gCoord = local2globalcoord([30; 45; 4],'sr');
% 30 degree azimuth, 45 degree elevation, 4 meter radius
```

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.

See Also

`global2localcoord` | `uv2azel` | `phitheta2azel` | `azel2uv` | `azel2phitheta`

Concepts

- "Global and Local Coordinate Systems"

Purpose

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 degrees kelvin.

Input Arguments

NBW

The noise bandwidth of the receiver in hertz. For a superheterodyne receiver, the noise bandwidth is approximately equal to the bandwidth of the intermediate frequency stages [1].

NF

Noise figure. 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 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 one.

REFTEMP

Reference temperature in degrees kelvin. The temperature of the receiver. Typical values range from 290–300 degrees kelvin.

Output Arguments

NPOWER

Noise power in watts. The internal noise power contribution of the receiver to the signal-to-noise ratio.

Examples

Calculate the noise power of a receiver whose noise bandwidth is 10 kHz, noise figure is 1 dB, and reference temperature is 300 K.

```
npower = noisepow(10e3,1,300);
```

References

[1] Skolnik, M. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

See Also

`phased.ReceiverPreamp` |

Purpose

Detection SNR threshold for signal in white Gaussian noise

Syntax

```
SNRTHRESH = npwgnthresh(PFA)
SNRTHRESH = npwgnthresh(PFA,NPULS)
SNRTHRESH = npwgnthresh(PFA,NPULS,DTYPE)
```

Description

`SNRTHRESH = npwgnthresh(PFA)` calculates the SNR threshold 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,NPULS)` specifies `NPULS` as the number of pulses used in the pulse integration.

`SNRTHRESH = npwgnthresh(PFA,NPULS,DTYPE)` specifies `DTYPE` as the type of detection. A square law detector is used in noncoherent detection.

Input Arguments

PFA

Probability of false alarm.

NPULS

Number of pulses used in the integration.

Default: `1`

DTYPE

Detection type.

Specify the type of pulse integration used in the NP decision rule. Valid choices for `DTYPE` are `'coherent'`, `'noncoherent'`, and `'real'`. `'coherent'` uses magnitude and phase information of complex-valued samples. `'noncoherent'` uses squared magnitudes. `'real'` uses real-valued samples.

Default: `'noncoherent'`

Output Arguments

SNRTHRESH

Signal-to-noise ratio threshold in decibels.

Definitions

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 the random variables are independent and identically distributed, with zero mean.

The threshold, λ, for an NP detector can be expressed as a signal-to-noise ratio in decibels:

$$10\log_{10}(\frac{\lambda^2}{\sigma^2}) = 10\log_{10}(2N(erfc^{-1}(2P_{FA}))^2)$$

In this equation:

- 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

Detection in Complex-Valued White Gaussian Noise (Coherent Samples)

The NP detector for complex-valued signals is similar to that discussed in “Detection in Real-Valued White Gaussian Noise” on page 4-82. 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 threshold for an NP detector expressed as a signal-to-noise ratio in decibels is:

$$10\log_{10}\left(\frac{\lambda^2}{\sigma^2}\right) = 10\log_{10}\left(N(erfc^{-1}(2P_{FA}))^2\right)$$

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 threshold for an NP detector expressed as a signal-to-noise ratio in decibels is:

```
10 * log10(gammaincinv(1-Pfa,npulses))
```

In this case, `gammaincinv` is the inverse of the incomplete gamma function, `Pfa` is the probability of false alarm, and `npulses` is the number of pulses.

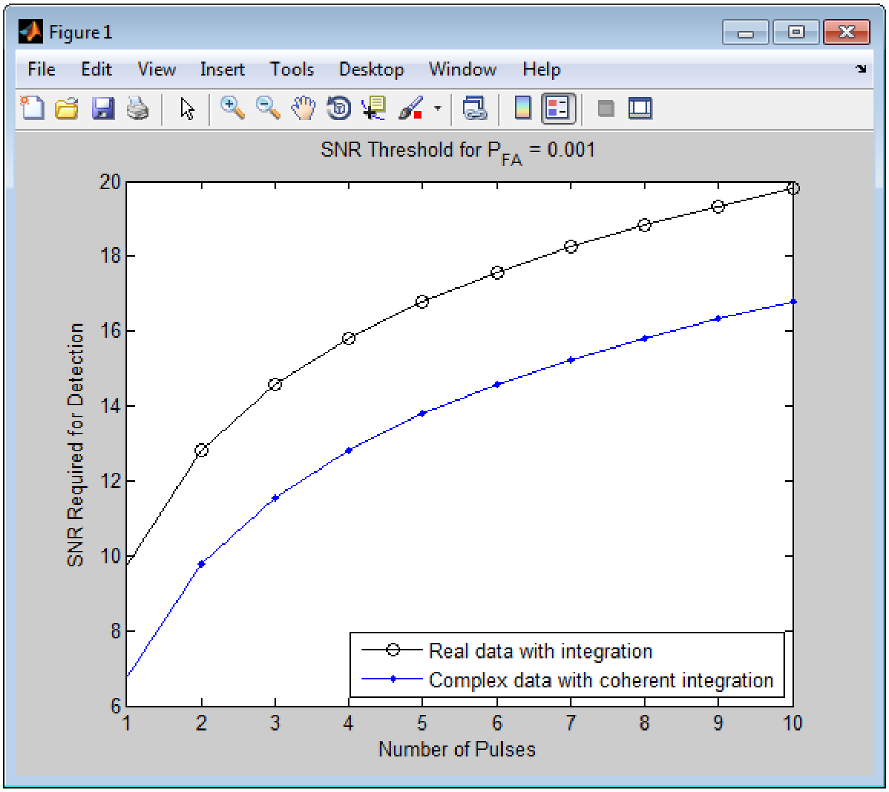
Examples

Calculate the SNR threshold that achieves a probability of false alarm 0.01 using a detection type of `'real'` with a single pulse. Then, verify that this threshold is producing a Pfa of approximately 0.01. Do so by constructing 10000 white real Gaussian noise samples and counting how many times the sample passes the threshold.

```
snrthreshold = npwgnthresh(0.01,1,'real');
npower = 1; Ntrial = 10000;
noise = sqrt(npower)*randn(1,Ntrial);
threshold = sqrt(npower*db2pow(snrthreshold));
calculated_Pfa = sum(noise>threshold)/Ntrial;
```

Plot the SNR threshold against the number of pulses, for real and complex data. In each case, the SNR threshold achieves a probability of false alarm of 0.001.

```
snrcoh = zeros(1,10); % Preallocate space
snrreal = zeros(1,10);
Pfa = 1e-3;
for num = 1:10
    snrreal(num) = npwgnthresh(Pfa,num,'real');
    snrcoh(num)  = npwgnthresh(Pfa,num,'coherent');
end
plot(snrreal,'ko-'); hold on;
plot(snrcoh,'b.-');
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 = 0.001')
hold off
```



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.

See Also

`rocpfa` | `rocsnr`

phitheta2azel

Purpose

Convert angles from phi/theta form to azimuth/elevation form

Syntax

`AzEl = phitheta2azel(PhiTheta)`

Description

`AzEl = phitheta2azel(PhiTheta)` converts the phi/theta angle pairs to their corresponding azimuth/elevation angle pairs.

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

AzEl - Azimuth/elevation angle pairs

two-row matrix

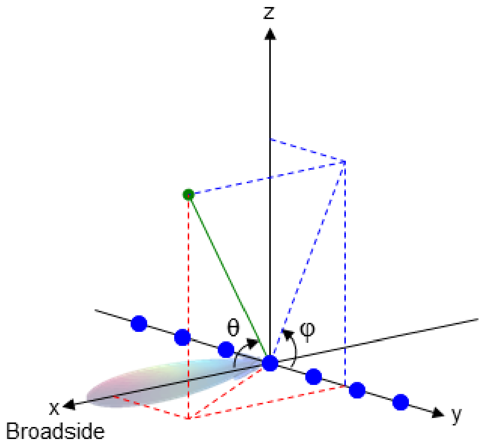
Azimuth and elevation angles, returned as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [azimuth; elevation]. The matrix dimensions of `AzEl` are the same as those of `PhiTheta`.

Definitions

Phi Angle, Theta Angle

The φ angle is the angle from the positive y-axis toward the positive z-axis, to the vector's orthogonal projection onto the yz plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the x-axis toward the yz plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.

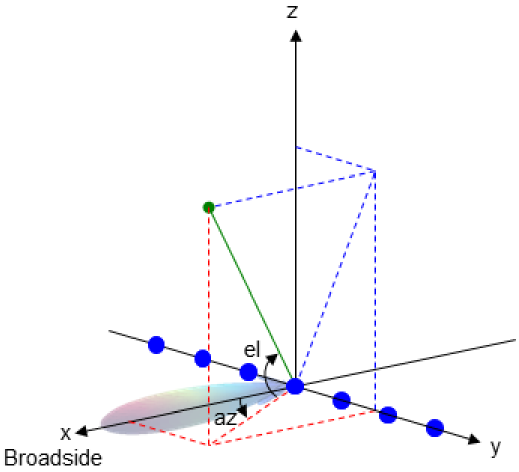


Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive *x*-axis toward the positive *y*-axis, to the vector's orthogonal projection onto the *xy* plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the *xy* plane toward the positive *z*-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of Phi/Theta Pair

Find the corresponding azimuth/elevation representation for $\varphi = 30$ degrees and $\theta = 0$ degrees.

```
AzEl = phitheta2azel([30; 0]);
```

See Also

`azel2phitheta`

Concepts

- “Spherical Coordinates”

Purpose

Convert radiation pattern from phi/theta form to azimuth/elevation form

Syntax

```
pat_azel = phitheta2azelpat(pat_phitheta,phi,theta)
pat_azel = phitheta2azelpat(pat_phitheta,phi,theta,az,el)
[pat_azel,az,el] = phitheta2azelpat( ___ )
```

Description

`pat_azel = phitheta2azelpat(pat_phitheta,phi,theta)` expresses the antenna radiation pattern `pat_phitheta` in azimuth/elevation angle coordinates instead of φ/θ angle coordinates. `pat_phitheta` samples the pattern at φ angles in `phi` and θ angles in `theta`. 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 = 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,az,el] = phitheta2azelpat( ___ )` 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.

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 360.

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 180.

Data Types
double

az - Azimuth angles

`[-180:180]` (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 –180 and 180.

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 form

M-by-L matrix

Antenna radiation pattern in azimuth/elevation form, returned as an M-by-L matrix. `pat_azel` samples the 3-D magnitude pattern in decibels, in terms of azimuth and elevation angles. L is the length of the `az` vector, and M is the length of the `el` vector.

az - Azimuth angles

vector of length L

Azimuth angles at which `pat_azel` samples the pattern, returned as a vector of length L. Angles are expressed in degrees.

el - Elevation angles

vector of length M

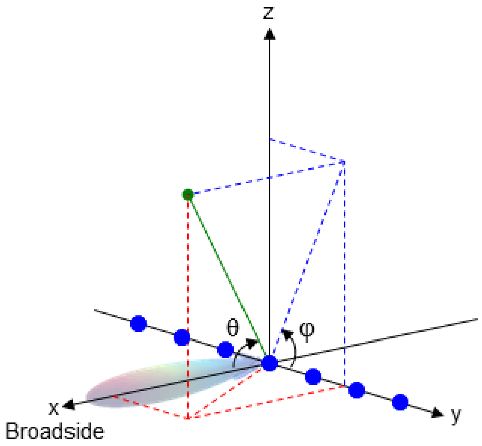
Elevation angles at which `pat_azel` samples the pattern, returned as a vector of length M. Angles are expressed in degrees.

Definitions

Phi Angle, Theta Angle

The φ angle is the angle from the positive *y*-axis toward the positive *z*-axis, to the vector's orthogonal projection onto the *yz* plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the *x*-axis toward the *yz* plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.

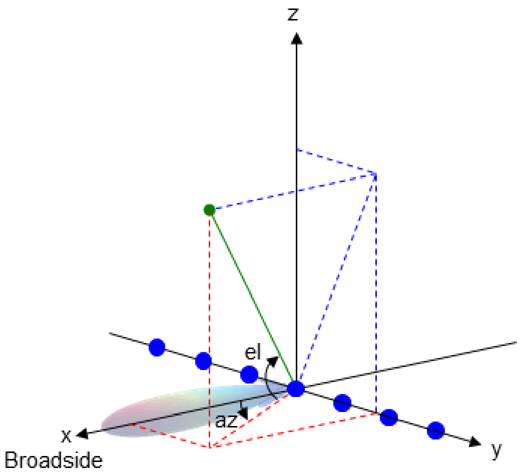


Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive *x*-axis toward the positive *y*-axis, to the vector's orthogonal projection onto the *xy* plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the *xy* plane toward the positive *z*-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of Radiation Pattern

Convert a radiation pattern to azimuth/elevation form, with the azimuth and elevation angles spaced 1 degree 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 of Converted Radiation Pattern

Convert a radiation pattern to azimuth/elevation form, with the azimuth and elevation angles spaced 1 degree apart.

Define the pattern in terms of φ and θ.

```
phi = 0:360;
```

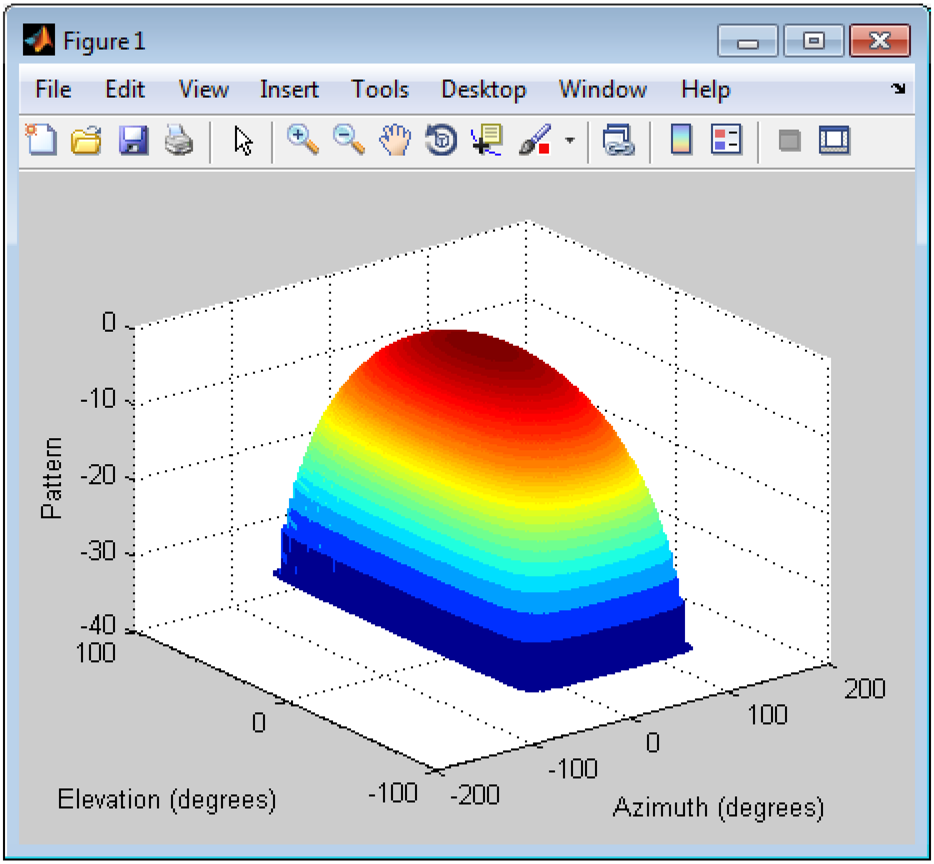
```
theta = 0:180;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Convert the pattern to azimuth/elevation space. Store the azimuth and elevation angles to use them for plotting.

```
[pat_azel,az,el] = phitheta2azelpat(pat_phitheta,phi,theta);
```

Plot the result.

```
H = surf(az,el,pat_azel);
set(H,'LineStyle','none')
xlabel('Azimuth (degrees)');
ylabel('Elevation (degrees)');
zlabel('Pattern');
```



Conversion of Radiation Pattern Using Specific Azimuth/Elevation Values

Convert a radiation pattern to azimuth/elevation form, with the azimuth and elevation angles spaced 5 degrees apart.

Define the pattern in terms of φ and θ.

```
phi = 0:360;
theta = 0:180;
```

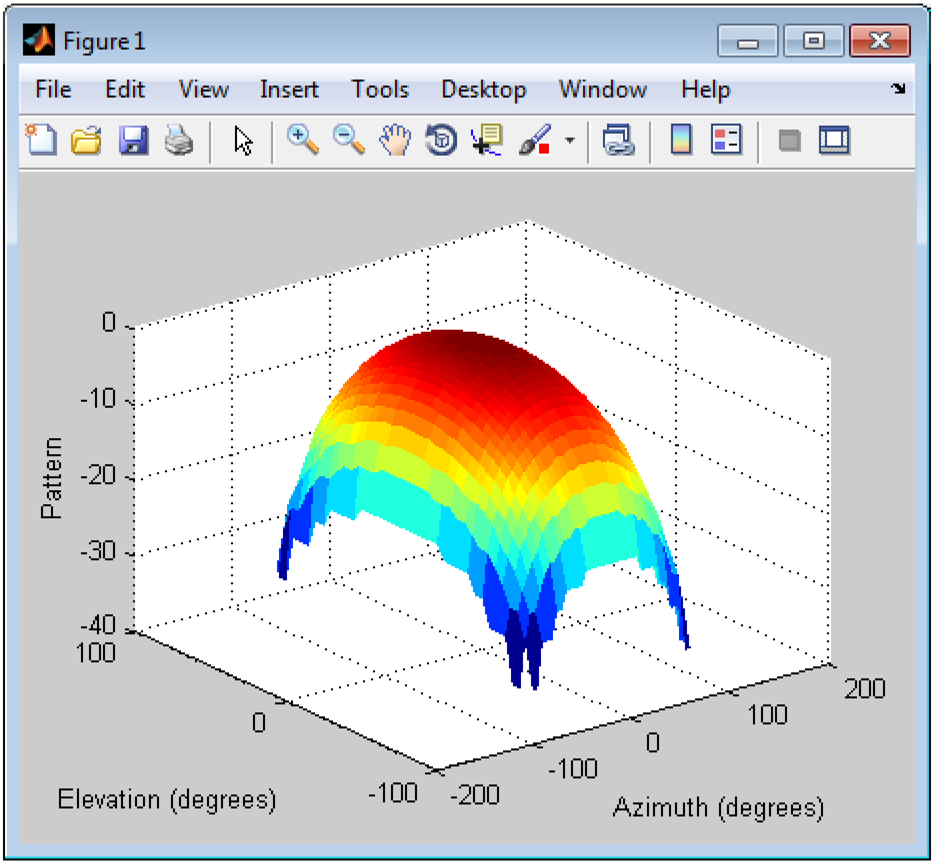
```
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

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 result.

```
H = surf(az,el,pat_azel);
set(H,'LineStyle','none')
xlabel('Azimuth (degrees)');
ylabel('Elevation (degrees)');
zlabel('Pattern');
```



See Also

`phased.CustomAntennaElement` **|** `phitheta2azel` | `azel2phitheta` | `azel2phithetapat`

Related Examples

- Antenna Array Analysis with Custom Radiation Pattern

Concepts

- “Spherical Coordinates”

phitheta2uv

Purpose

Convert phi/theta angles to u/v coordinates

Syntax

```
UV = phitheta2uv(PhiTheta)
```

Description

`UV = phitheta2uv(PhiTheta)` converts the phi/theta angle pairs to their corresponding *u*/*v* space coordinates.

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

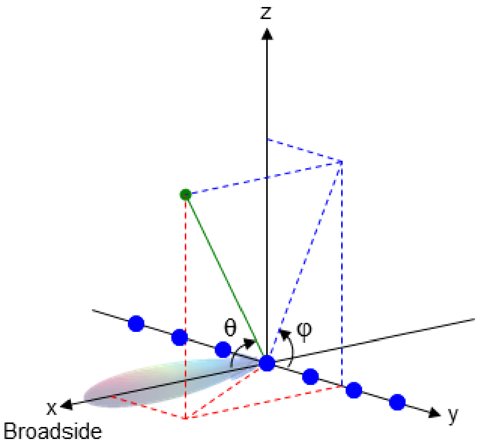
Angle in *u*/*v* space, returned as a two-row matrix. Each column of the matrix represents an angle in the form [*u*; *v*]. The matrix dimensions of `UV` are the same as those of `PhiTheta`.

Definitions

Phi Angle, Theta Angle

The φ angle is the angle from the positive *y*-axis toward the positive *z*-axis, to the vector's orthogonal projection onto the *yz* plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the *x*-axis toward the *yz* plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



U/V Space

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, as follows:

$$u = \sin(\theta)\cos(\varphi)$$

$$v = \sin(\theta)\sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

The values of *u* and *v* satisfy these inequalities:

$$-1 \leq u \leq 1$$

$$-1 \leq v \leq 1$$

$$u^2 + v^2 \leq 1$$

Examples

Conversion of Phi/Theta Pair

Find the corresponding *u*/*v* representation for $\varphi = 30$ degrees and $\theta = 0$ degrees.

```
UV = phitheta2uv([30; 0]);
```

See Also `uv2phitheta`

Concepts

- “Spherical Coordinates”

Purpose

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,v] = phitheta2uvpat( ___ )
```

Description

`pat_uv = phitheta2uvpat(pat_phitheta,phi,theta)` expresses the antenna radiation pattern `pat_phitheta` in u/v space coordinates instead of φ/θ angle 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,v] = 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.

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 - *u* coordinates

vector of length L

u coordinates at which `pat_uv` samples the pattern, returned as a vector of length L.

v - *v* coordinates

vector of length M

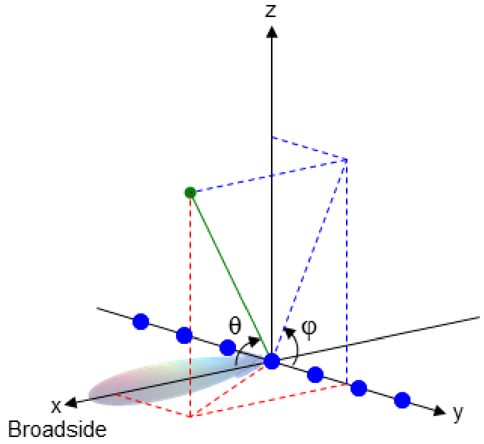
v coordinates at which `pat_uv` samples the pattern, returned as a vector of length M.

Definitions

Phi Angle, Theta Angle

The φ angle is the angle from the positive *y*-axis toward the positive *z*-axis, to the vector's orthogonal projection onto the *yz* plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the *x*-axis toward the *yz* plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



U/V Space

The u/v coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, as follows:

$$u = \sin(\theta)\cos(\varphi)$$

$$v = \sin(\theta)\sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

The values of u and v satisfy these inequalities:

$$-1 \leq u \leq 1$$

$$-1 \leq v \leq 1$$

$$u^2 + v^2 \leq 1$$

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* space.

```
pat_uv = phitheta2uvpat(pat_phitheta,phi,theta);
```

Plot of Converted 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.

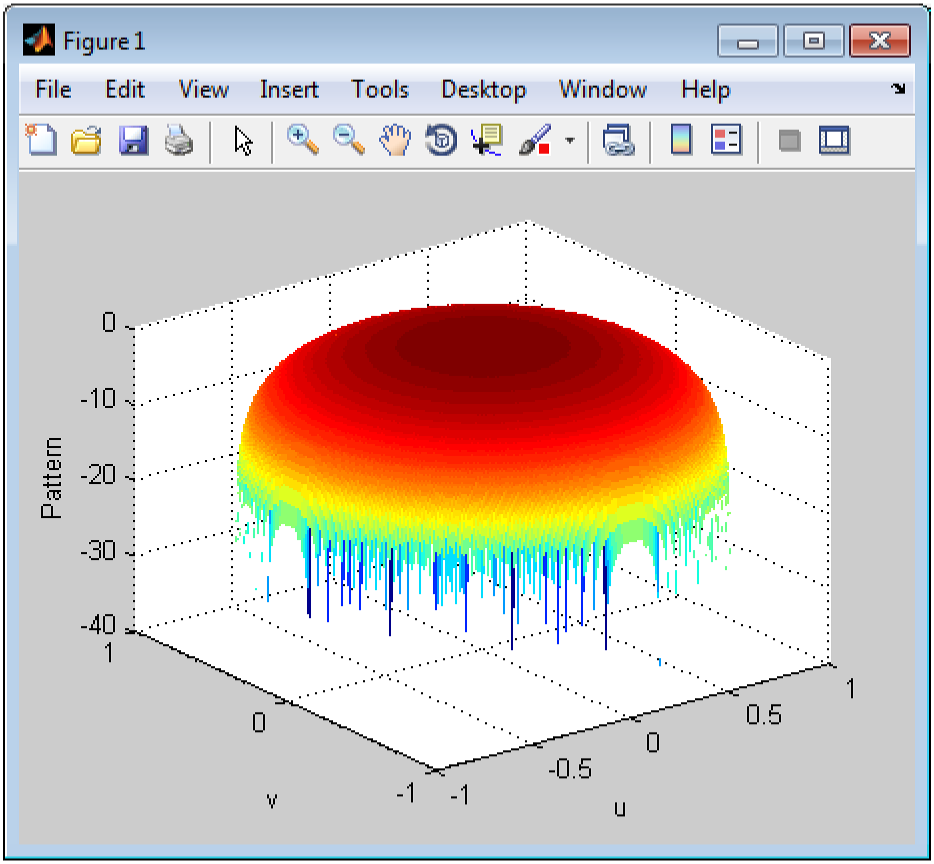
```
phi = 0:360;
theta = 0:90;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Convert the pattern to *u*/*v* space. Store the *u* and *v* coordinates to use them for plotting.

```
[pat_uv,u,v] = phitheta2uvpat(pat_phitheta,phi,theta);
```

Plot the result.

```
H = surf(u,v,pat_uv);
set(H,'LineStyle','none')
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



Conversion of Radiation Pattern Using Specific U/V Values

Convert a radiation pattern to *u*/*v* form, with the *u* and *v* coordinates spaced by 0.05.

Define the pattern in terms of φ and θ.

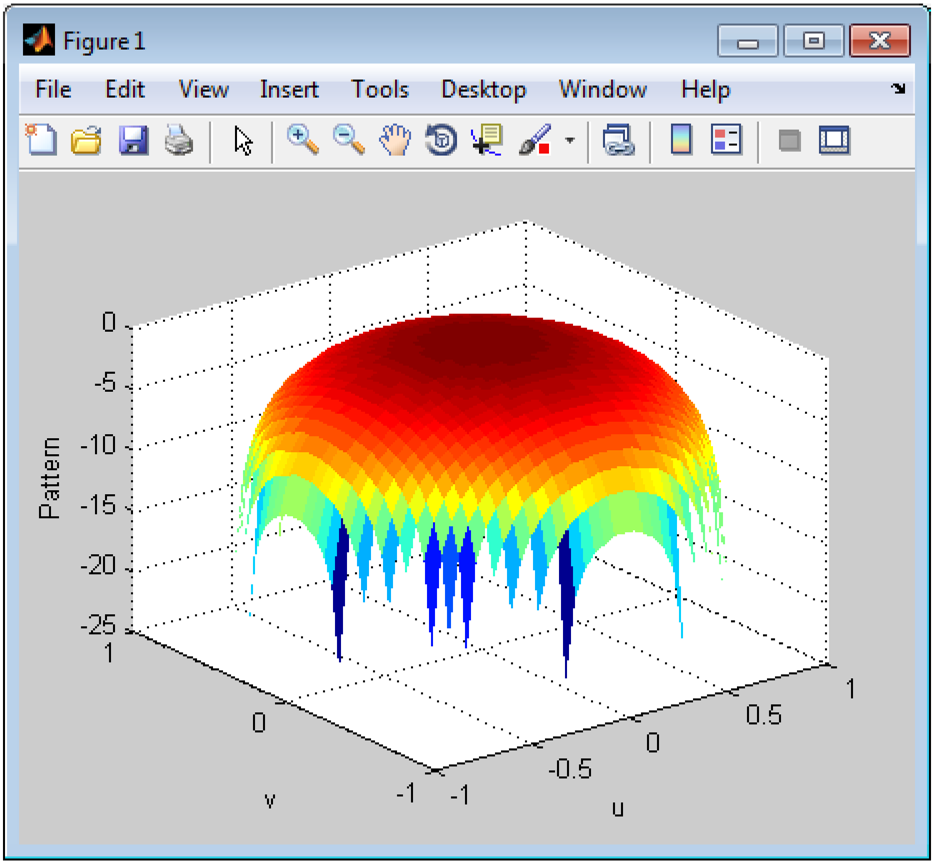
```
phi = 0:360;
theta = 0:90;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

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);
set(H,'LineStyle','none')
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



See Also

`phased.CustomAntennaElement` | `phitheta2uv` | `uv2phitheta` | `uv2phithetapat`

Concepts

- "Spherical Coordinates"

Purpose

Physical constants

Syntax

```
Const = physconst(Name)
```

Description

`Const = physconst(Name)` returns the constant corresponding to the string `Name` in SI units. Valid values of `Name` are `'LightSpeed'`, `'Boltzmann'`, and `'EarthRadius'`.

Input Arguments

Name

String that indicates which physical constant the function returns. The valid strings are not case sensitive.

Definitions

The following table lists the supported constants and their values in SI units.

Constant	Description	Value
`'LightSpeed'`	Speed of light in a vacuum	299,792,458 m/s. Most commonly denoted by *c*.
`'Boltzmann'`	Boltzmann constant relating energy to temperature	$1.38\text{x}10^{-23}$ J/K. Most commonly denoted by *k*.
`'EarthRadius'`	Mean radius of the Earth	6,371,000 m

Examples

Wavelength Corresponding to Known Frequency

Determine the wavelength of an electromagnetic wave whose frequency is 1 GHz.

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

Thermal Noise Power

Approximate the thermal noise power per unit bandwidth in the I and Q channels of a receiver.

Define the receiver temperature and Boltzmann constant.

```
T = 290;
k = physconst('Boltzmann');
```

Compute the noise power per unit bandwidth, split evenly between the in-phase and quadrature channels.

```
Noise_power = 10*log10(k*T/2);
```

Purpose

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'`.

Input Arguments

X

Pulse input data. Each column of `X` is one pulse.

METHOD

Pulse integration method. `METHOD` is the method used to integrate the pulses in the columns of `X`. Valid values of `METHOD` are `'coherent'` and `'noncoherent'`. The strings are not case sensitive.

Default: `'noncoherent'`

Output Arguments

Y

Integrated pulse. `Y` is an N-by-1 column vector where N is the number of rows in the input `X`.

Definitions

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.

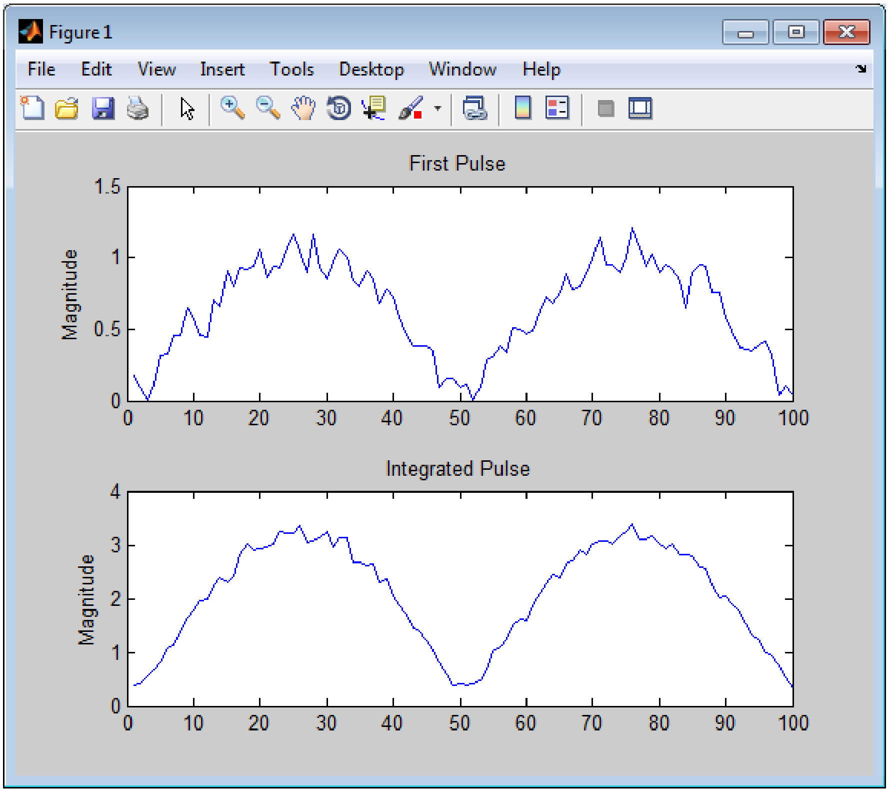
The noncoherent (video) integration of the pulses in X is:

$$Y_i = \sqrt{\sum_{j=1}^{N} |X_{ij}|^2}$$

Examples

Noncoherently integrate 10 pulses.

```
x = repmat(sin(2*pi*(0:99)'/100),1,10)+0.1*randn(100,10);
y = pulsint(x);
subplot(211), plot(abs(x(:,1)));
ylabel('Magnitude');
title('First Pulse');
subplot(212), plot(abs(y));
ylabel('Magnitude');
title('Integrated Pulse');
```



References

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

See Also

`phased.MatchedFilter` |

Purpose

Peak power estimate from radar equation

Syntax

```
Pt = radareqpow(lambda,tgtrng,SNR,Tau)
Pt = radareqpow(...,Name,Value)
```

Description

`Pt = radareqpow(lambda,tgtrng,SNR,Tau)` estimates the peak transmit power required for a radar operating at a wavelength of `lambda` meters to achieve the specified signal-to-noise ratio `SNR` in decibels for a target at a range of `tgtrng` meters. The target has a nonfluctuating radar cross section (RCS) of 1 square meter.

`Pt = radareqpow(...,Name,Value)` estimates the required peak transmit power with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

lambda

Wavelength of radar operating frequency (in meters). The wavelength is the ratio of the wave propagation speed to frequency. For electromagnetic waves, the speed of propagation is the speed of light. Denoting the speed of light by *c* and the frequency (in hertz) of the wave by *f*, the equation for wavelength is:

$$\lambda = \frac{c}{f}$$

tgtrng

Target range in meters. When the transmitter and receiver are colocated (monostatic radar), `tgtrng` is a real-valued positive scalar. When the transmitter and receiver are not colocated (bistatic radar), `tgtrng` is a 1-by-2 row vector with real-valued positive elements. The first element is the target range from the transmitter, and the second element is the target range from the receiver.

SNR

The minimum output signal-to-noise ratio at the receiver in decibels.

Tau

Single pulse duration in seconds.

Name-Value Pair Arguments

Gain

Transmitter and receiver gain in decibels (dB). When the transmitter and receiver are colocated (monostatic radar), `Gain` is a real-valued scalar. The transmit and receive gains are equal. When the transmitter and receiver are not colocated (bistatic radar), `Gain` is a 1-by-2 row vector with real-valued elements. The first element is the transmitter gain and the second element is the receiver gain.

Default: 20

Loss

System loss in decibels (dB). `Loss` represents a general loss factor that comprises losses incurred in the system components and in the propagation to and from the target.

Default: 0

RCS

Radar cross section in square meters. The target RCS is nonfluctuating.

Default: 1

Ts

System noise temperature in kelvin. The system noise temperature is the product of the system temperature and the noise figure.

Default: 290 kelvin

Output Arguments

Pt

Transmitter peak power in watts.

Definitions

Point Target Radar Range Equation

The point target radar range equation estimates the power at the input to the receiver for a target of a given radar cross section at a specified range. The model is deterministic and assumes isotropic radiators. The equation for the power at the input to the receiver is

$$P_r = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 R_t^2 R_r^2 L}$$

where the terms in the equation are:

- P_t — Peak transmit power in watts
- G_t — Transmitter gain in decibels
- G_r — Receiver gain in decibels. If the radar is monostatic, the transmitter and receiver gains are identical.
- λ — Radar operating frequency wavelength in meters
- σ — Target's nonfluctuating radar cross section in square meters
- L — General loss factor in decibels that accounts for both system and propagation loss
- R_t — Range from the transmitter to the target
- R_r — Range from the receiver to the target. If the radar is monostatic, the transmitter and receiver ranges are identical.

Terms expressed in decibels such as the loss and gain factors enter the equation in the form $10^{x/10}$ where x denotes the variable. For example, the default loss factor of 0 dB results in a loss term of $10^{0/10}=1$.

Receiver Output Noise Power

The equation for the power at the input to the receiver represents the *signal* term in the signal-to-noise ratio. To model the noise term, assume the thermal noise in the receiver has a white noise power spectral density (PSD) given by:

$$P(f) = kT$$

where k is the Boltzmann constant and T is the effective noise temperature. The receiver acts as a filter to shape the white noise PSD. Assume that the magnitude squared receiver frequency response approximates a rectangular filter with bandwidth equal to the reciprocal of the pulse duration, $1/\tau$. The total noise power at the output of the receiver is:

$$N = \frac{kTF_n}{\tau}$$

where F_n is the receiver *noise factor*.

The product of the effective noise temperature and the receiver noise factor is referred to as the *system temperature* and is denoted by T_s, so that $T_s = TF_n$.

Receiver Output SNR

Using the equation for the received signal power in "Point Target Radar Range Equation" on page 4-116 and the output noise power in "Receiver Output Noise Power" on page 4-116, the receiver output SNR is:

$$\frac{P_r}{N} = \frac{P_t \tau G_t G_r \lambda^2 \sigma}{(4\pi)^3 k T_s R_t^2 R_r^2 L}$$

Solving for the peak transmit power

$$P_t = \frac{P_r (4\pi)^3 k T_s R_t^2 R_r^2 L}{N \tau G_t G_r \lambda^2 \sigma}$$

Examples

Estimate the required peak transmit power required to achieve a minimum SNR of 6 decibels for a target at a range of 50 kilometers. The target has a nonfluctuating RCS of 1 square meter. The radar operating frequency is 1 gigahertz. The pulse duration is 1 microsecond.

```
lambda = physconst('LightSpeed')/1e9;
tgtrng = 50e3;
tau = 1e-6;
SNR = 6;
Pt = radareqpow(lambda,tgtrng,SNR,tau);
```

Estimate the required peak transmit power required to achieve a minimum SNR of 10 decibels for a target with an RCS of 0.5 square meters at a range of 50 kilometers. The radar operating frequency is 10 gigahertz. The pulse duration is 1 microsecond. Assume a transmit and receive gain of 30 decibels and an overall loss factor of 3 decibels.

```
lambda = physconst('LightSpeed')/10e9;
Pt = radareqpow(lambda,50e3,10,1e-6,'RCS',0.5,...
    'Gain',30,'Ts',300,'Loss',3);
```

Estimate the required peak transmit power for a bistatic radar to achieve a minimum SNR of 6 decibels for a target with an RCS of 1 square meter. The target is 50 kilometers from the transmitter and 75 kilometers from the receiver. The radar operating frequency is 10 gigahertz and the pulse duration is 10 microseconds. The transmitter and receiver gains are 40 and 20 dB respectively.

```
lambda = physconst('LightSpeed')/10e9;
SNR = 6;
tau = 10e-6;
TxRng = 50e3; RvRng = 75e3;
TxRvRng =[TxRng RvRng];
TxGain = 40; RvGain = 20;
Gain = [TxGain RvGain];
Pt = radareqpow(lambda,TxRvRng,SNR,tau,'Gain',Gain);
```

References

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

[2] Skolnik, M. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

[3] Willis, N. J. *Bistatic Radar*. Raleigh, NC: SciTech Publishing, 2005.

See Also

`phased.Transmitter` | `phased.ReceiverPreamp` | `noisepow` | `radareqrng` | `radareqsnr` | `systemp`

radareqrng

Purpose

Maximum theoretical range estimate

Syntax

```
maxrng = radareqrng(lambda,SNR,Pt,Tau)
maxrng = radareqrng(...,Name,Value)
```

Description

`maxrng = radareqrng(lambda,SNR,Pt,Tau)` estimates the theoretical maximum detectable range `maxrng` for a radar operating with a wavelength of `lambda` meters with a pulse duration of `Tau` seconds. The signal-to-noise ratio is `SNR` decibels, and the peak transmit power is `Pt` watts.

`maxrng = radareqrng(...,Name,Value)` estimates the theoretical maximum detectable range with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

lambda

Wavelength of radar operating frequency (in meters). The wavelength is the ratio of the wave propagation speed to frequency. For electromagnetic waves, the speed of propagation is the speed of light. Denoting the speed of light by *c* and the frequency (in hertz) of the wave by *f*, the equation for wavelength is:

$$\lambda = \frac{c}{f}$$

Pt

Transmitter peak power in watts.

SNR

The minimum output signal-to-noise ratio at the receiver in decibels.

Tau

Single pulse duration in seconds.

Name-Value Pair Arguments

Gain

Transmitter and receiver gain in decibels (dB). When the transmitter and receiver are colocated (monostatic radar), `Gain` is a real-valued scalar. The transmit and receive gains are equal. When the transmitter and receiver are not colocated (bistatic radar), `Gain` is a 1-by-2 row vector with real-valued elements. The first element is the transmitter gain, and the second element is the receiver gain.

Default: 20

Loss

System loss in decibels (dB). `Loss` represents a general loss factor that comprises losses incurred in the system components and in the propagation to and from the target.

Default: 0

RCS

Radar cross section in square meters. The target RCS is nonfluctuating.

Default: 1

Ts

System noise temperature in kelvins. The system noise temperature is the product of the system temperature and the noise figure.

Default: 290 kelvin

unitstr

The units of the estimated maximum theoretical range. `unitstr` is one of the following strings:

- `'km'` kilometers

- `'m'` meters
- `'nmi'` nautical miles (U.S.)

Default: `'m'`

Output Arguments

maxrng

The estimated theoretical maximum detectable range. The units of `maxrng` depends on the value of `unitstr`. By default `maxrng` is in meters. For bistatic radars, `maxrng` is the geometric mean of the range from the transmitter to the target and the receiver to the target.

Definitions

Point Target Radar Range Equation

The point target radar range equation estimates the power at the input to the receiver for a target of a given radar cross section at a specified range. The model is deterministic and assumes isotropic radiators. The equation for the power at the input to the receiver is

$$P_r = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 R_t^2 R_r^2 L}$$

where the terms in the equation are:

- P_t — Peak transmit power in watts
- G_t — Transmitter gain in decibels
- G_r — Receiver gain in decibels. If the radar is monostatic, the transmitter and receiver gains are identical.
- λ — Radar operating frequency wavelength in meters
- σ — Target's nonfluctuating radar cross section in square meters
- L — General loss factor in decibels that accounts for both system and propagation loss
- R_t — Range from the transmitter to the target

- R_r — Range from the receiver to the target. If the radar is monostatic, the transmitter and receiver ranges are identical.

Terms expressed in decibels, such as the loss and gain factors, enter the equation in the form $10^{x/10}$ where x denotes the variable. For example, the default loss factor of 0 dB results in a loss term of $10^{0/10}=1$.

Receiver Output Noise Power

The equation for the power at the input to the receiver represents the *signal* term in the signal-to-noise ratio. To model the noise term, assume the thermal noise in the receiver has a white noise power spectral density (PSD) given by:

$$P(f) = kT$$

where k is the Boltzmann constant and T is the effective noise temperature. The receiver acts as a filter to shape the white noise PSD. Assume that the magnitude squared receiver frequency response approximates a rectangular filter with bandwidth equal to the reciprocal of the pulse duration, $1/\tau$. The total noise power at the output of the receiver is:

$$N = \frac{kTF_n}{\tau}$$

where F_n is the receiver *noise factor*.

The product of the effective noise temperature and the receiver noise factor is referred to as the *system temperature*. This value is denoted by T_s, so that $T_s=TF_n$.

Receiver Output SNR

The receiver output SNR is:

$$\frac{P_r}{N} = \frac{P_t \tau G_t G_r \lambda^2 \sigma}{(4\pi)^3 k T_s R_t^2 R_r^2 L}$$

You can derive this expression using the following equations:

- Received signal power in “Point Target Radar Range Equation” on page 4-122
- Output noise power in “Receiver Output Noise Power” on page 4-123

Theoretical Maximum Detectable Range

For monostatic radars, the range from the target to the transmitter and receiver is identical. Denoting this range by R, you can express this relationship as $R^4 = R_t^2 R_r^2$.

Solving for R

$$R = \left(\frac{NP_t \tau G_t G_r \lambda^2 \sigma}{P_r (4\pi)^3 k T_s L}\right)^{1/4}$$

For bistatic radars, the theoretical maximum detectable range is the geometric mean of the ranges from the target to the transmitter and receiver:

$$\sqrt{R_t R_r} = \left(\frac{NP_t \tau G_t G_r \lambda^2 \sigma}{P_r (4\pi)^3 k T_s L}\right)^{1/4}$$

Examples

Estimate the theoretical maximum detectable range for a monostatic radar operating at 10 GHz using a pulse duration of 10 μs. Assume the output SNR of the receiver is 6 dB.

```
lambda = physconst('LightSpeed')/10e9;
SNR = 6;
tau = 10e-6;
Pt = 1e6;
maxrng = radareqrng(lambda,SNR,Pt,tau);
```

Estimate the theoretical maximum detectable range for a monostatic radar operating at 10 GHz using a pulse duration of 10 μs. The target

RCS is 0.1 square meters. Assume the output SNR of the receiver is 6 dB. The transmitter-receiver gain is 40 dB. Assume a loss factor of 3 dB.

```
lambda = physconst('LightSpeed')/10e9;
SNR = 6;
tau = 10e-6;
Pt = 1e6;
RCS = 0.1;
Gain = 40;
Loss = 3;
maxrng2 = radareqrng(lambda,SNR,Pt,tau,'Gain',Gain,...
    'RCS',RCS,'Loss',Loss);
```

References

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

[2] Skolnik, M. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

[3] Willis, N. J. *Bistatic Radar*. Raleigh, NC: SciTech Publishing, 2005.

See Also

`phased.Transmitter` | `phased.ReceiverPreamp` | `noisepow` | `radareqpow` | `radareqsnr` | `systemp`

radareqsnr

Purpose

SNR estimate from radar equation

Syntax

```
SNR = radareqsnr(lambda,tgtrng,Pt,tau)
SNR = radareqsnr(...,Name,Value)
```

Description

`SNR = radareqsnr(lambda,tgtrng,Pt,tau)` estimates the output signal-to-noise ratio (SNR) at the receiver based on the wavelength `lambda` in meters, the range `tgtrng` in meters, the peak transmit power `Pt` in watts, and the pulse width `tau` in seconds.

`SNR = radareqsnr(...,Name,Value)` estimates the output SNR at the receiver with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

lambda

Wavelength of radar operating frequency in meters. The wavelength is the ratio of the wave propagation speed to frequency. For electromagnetic waves, the speed of propagation is the speed of light. Denoting the speed of light by *c* and the frequency in hertz of the wave by *f*, the equation for wavelength is:

$$\lambda = \frac{c}{f}$$

tgtrng

Target range in meters. When the transmitter and receiver are colocated (monostatic radar), `tgtrng` is a real-valued positive scalar. When the transmitter and receiver are not colocated (bistatic radar), `tgtrng` is a 1-by-2 row vector with real-valued positive elements. The first element is the target range from the transmitter, and the second element is the target range from the receiver.

Pt

Transmitter peak power in watts.

tau

Single pulse duration in seconds.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Gain

Transmitter and receiver gain in decibels (dB). When the transmitter and receiver are colocated (monostatic radar), `Gain` is a real-valued scalar. The transmit and receive gains are equal. When the transmitter and receiver are not colocated (bistatic radar), `Gain` is a 1-by-2 row vector with real-valued elements. The first element is the transmitter gain, and the second element is the receiver gain.

Default: 20

Loss

System loss in decibels (dB). `Loss` represents a general loss factor that comprises losses incurred in the system components and in the propagation to and from the target.

Default: 0

RCS

Target radar cross section in square meters. The target RCS is nonfluctuating.

Default: 1

Ts

System noise temperature in kelvin. The system noise temperature is the product of the effective noise temperature and the noise figure.

Default: 290 kelvin

Output Arguments

SNR

The estimated output signal-to-noise ratio at the receiver in decibels. `SNR` is $10\log_{10}(P_r/N)$. The ratio P_r/N is defined in "Receiver Output SNR" on page 4-129.

Definitions

Point Target Radar Range Equation

The point target radar range equation estimates the power at the input to the receiver for a target of a given radar cross section at a specified range. The model is deterministic and assumes isotropic radiators. The equation for the power at the input to the receiver is

$$P_r = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 R_t^2 R_r^2 L}$$

where the terms in the equation are:

- P_t — Peak transmit power in watts
- G_t — Transmitter gain in decibels
- G_r — Receiver gain in decibels. If the radar is monostatic, the transmitter and receiver gains are identical.
- λ — Radar operating frequency wavelength in meters
- σ — Nonfluctuating target radar cross section in square meters
- L — General loss factor in decibels that accounts for both system and propagation losses
- R_t — Range from the transmitter to the target in meters
- R_r — Range from the receiver to the target in meters. If the radar is monostatic, the transmitter and receiver ranges are identical.

Terms expressed in decibels such as the loss and gain factors enter the equation in the form $10^{x/10}$ where x denotes the variable value in decibels. For example, the default loss factor of 0 dB results in a loss term equal to one in the equation ($10^{0/10}$).

Receiver Output Noise Power

The equation for the power at the input to the receiver represents the signal term in the signal-to-noise ratio. To model the noise term, assume the thermal noise in the receiver has a white noise power spectral density (PSD) given by:

$$P(f) = kT$$

where k is the Boltzmann constant and T is the effective noise temperature. The receiver acts as a filter to shape the white noise PSD. Assume that the magnitude squared receiver frequency response approximates a rectangular filter with bandwidth equal to the reciprocal of the pulse duration, $1/\tau$. The total noise power at the output of the receiver is:

$$N = \frac{kTF_n}{\tau}$$

where F_n is the receiver *noise factor*.

The product of the effective noise temperature and the receiver noise factor is referred to as the *system temperature* and is denoted by T_s, so that $T_s = TF_n$.

Receiver Output SNR

The receiver output SNR is:

$$\frac{P_r}{N} = \frac{P_t \tau G_t G_r \lambda^2 \sigma}{(4\pi)^3 k T_s R_t^2 R_r^2 L}$$

You can derive this expression using the following equations:

- Received signal power in “Point Target Radar Range Equation” on page 4-128

- Output noise power in "Receiver Output Noise Power" on page 4-129

Examples

Estimate the output SNR for a target with an RCS of 1 square meter at a range of 50 kilometers. The system is a monostatic radar operating at 1 gigahertz with a peak transmit power of 1 megawatt and pulse width of 0.2 microseconds. The transmitter and receiver gain is 20 decibels and the system temperature is 290 kelvin.

```
lambda = physconst('LightSpeed')/1e9;
tgtrng = 50e3;
Pt = 1e6;
tau = 0.2e-6;
snr = radareqsnr(lambda,tgtrng,Pt,tau);
```

Estimate the output SNR for a target with an RCS of 0.5 square meters at 100 kilometers. The system is a monostatic radar operating at 10 gigahertz with a peak transmit power of 1 megawatt and pulse width of 1 microsecond. The transmitter and receiver gain is 40 decibels. The system temperature is 300 kelvin and the loss factor is 3 decibels.

```
lambda = physconst('LightSpeed')/10e9;
snr = radareqsnr(lambda,100e3,1e6,1e-6,'RCS',0.5,...
    'Gain',40,'Ts',300,'Loss',3);
```

Estimate the output SNR for a target with an RCS of 1 square meter. The radar is bistatic. The target is located 50 kilometers from the transmitter and 75 kilometers from the receiver. The radar operating frequency is 10 gigahertz. The transmitter has a peak transmit power of 1 megawatt with a gain of 40 decibels. The pulse width is 1 microsecond. The receiver gain is 20 decibels.

```
lambda = physconst('LightSpeed')/10e9;
tau = 1e-6;
Pt = 1e6;
txrvRng =[50e3 75e3];
```

```
Gain = [40 20];
snr = radareqsnr(lambda,txrvRng,Pt,tau,'Gain',Gain);
```

References

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

[2] Skolnik, M. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

[3] Willis, N. J. *Bistatic Radar*. Raleigh, NC: SciTech Publishing, 2005.

See Also

`phased.Transmitter` **|** `phased.ReceiverPreamp` **|** `noisepow` | `radareqrng` | `radareqpow` | `systemp`

radialspeed

Purpose

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.

Input Arguments

Pos

Positions of platforms, specified as a 3-by-N matrix. Each column specifies a position in the form [*x*; *y*; *z*], in meters.

V

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.

RefPos

Position of reference platform, specified as a 3-by-1 vector. The vector has the form [*x*; *y*; *z*], in meters.

Default: `[0; 0; 0]`

RefV

Velocity of reference platform, specified as a 3-by-1 vector. The vector has the form [*x*; *y*; *z*], in meters per second.

Default: `[0; 0; 0]`

Output Arguments

Rspeed

Radial speed in meters per second, 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.

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 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]);
```

See Also

`phased.Platform` | `speed2dop`

Concepts

- "Doppler Shift and Pulse-Doppler Processing"
- "Motion Modeling in Phased Array Systems"

range2beat

Purpose

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. `slope` is the slope of the FMCW sweep.

`fb = range2beat(r,slope,c)` specifies the signal propagation speed.

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

Definitions

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`.

Examples

Maximum Beat Frequency in FMCW Radar System

Calculate the maximum beat frequency in the received signal of an upsweep FMCW waveform. Assume that the waveform can detect a target as far as 18 km and sweeps a 300 MHz band in 1 ms. Also assume that the target is stationary.

```
slope = 300e6/1e-3;
r = 18e3;
fb = range2beat(r,slope);
```

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.

See Also

`beat2range` | `dechirp` | `rdcoupling` | `stretchfreq2rng` `phased.FMCWWaveform` |

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

Purpose

Convert range resolution to required bandwidth

Syntax

```
bw = range2bw(r)
bw = range2bw(r,c)
```

Description

`bw = range2bw(r)` 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(r,c)` specifies the signal propagation speed.

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.

Input Arguments

r - Target range resolution

array of positive numbers

Target range resolution in meters, specified as an array of positive 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

bw - Required bandwidth

array of nonnegative numbers

Required bandwidth in hertz, returned as an array of nonnegative numbers. The dimensions of `bw`are the same as those of `r`.

Algorithms

The function computes `c/(2*r)`.

Examples

Pulse Width for Specified Range Resolution

Assume you have a monostatic radar system that uses a rectangular waveform. Calculate the required pulse width of the waveform so that the system can achieve a range resolution of 10 m.

```
r = 10;
tau = 1/range2bw(r);
```

References

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

See Also

`time2range | range2timephased.FMCWWaveform` **|**

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

Purpose

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.

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`.

Examples

PRF for Specified Unambiguous Range

Calculate the required PRF for a monostatic radar system so that it can have a maximum unambiguous range of 15 km.

```
r = 15e3;
prf = 1/range2time(r);
```

References

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

See Also

`time2range` | `range2bw` `phased.FMCWWaveform` |

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

Purpose

Range and angle calculation

Syntax

```
[tgtrng,tgtang] = rangeangle(POS)
[tgtrng,tgtang] = rangeangle(POS,REFPOS)
[tgtrng,tgtang] = rangeangle(POS,REFPOS,REFAXES)
```

Description

`[tgtrng,tgtang] = rangeangle(POS)` returns the range, `tgtrng`, and direction, `tgtang`, from the origin to the position, `POS`.

`[tgtrng,tgtang] = rangeangle(POS,REFPOS)` returns the range and angle from the reference position, `REFPOS`, to the position `POS`.

`[tgtrng,tgtang] = rangeangle(POS,REFPOS,REFAXES)` returns the range and angle of `POS` in the local coordinate system whose origin is `REFPOS` and whose axes are defined in `REFAXES`.

Input Arguments

POS

Input position in meters. `POS` is 3-by-N matrix of rectangular coordinates in the form `[x;y;z]`. Each column in `POS` represents the coordinates of one position.

REFPOS

Reference position. `REFPOS` is a 3-by-1 vector of rectangular coordinates in the form `[x;y;z]`. `REFPOS` serves as the origin of the local coordinate system. Ranges and angles to the columns of `POS` are measured with respect to `REFPOS`.

Default: `[0;0;0]`

REFAXES

Local coordinate system axes. `REFAXES` is a 3-by-3 matrix whose columns define the axes the of the local coordinate system with origin at `REFPOS`. Each column in `REFAXES` specifies the direction of an axis for the local coordinate system in rectangular coordinates `[x; y; z]`.

Default: `[0 1 0;0 0 1;1 0 0]`

Output Arguments

tgtrng

Range in meters. `tgtrng` is an 1-by-N vector of ranges from the origin to the corresponding columns in `POS`.

tgtang

Azimuth and elevation angles in degrees. `tgtang` is a 2-by-N matrix whose columns are the angles in the form `[azimuth;elevation]` for the corresponding positions specified in `POS`.

Examples

Find the range and angle of a target located at (1000,2000,50).

```
TargetLoc = [1e3;2e3;50];
[tgtrng,tgtang] = rangeangle(TargetLoc);
```

Find the range and angle of a target located at (1000,2000,50) with respect to a local origin at (100,100,10).

```
TargetLoc = [1e3;2e3;50];
[tgtrng,tgtang] = rangeangle(TargetLoc,[100; 100; 10]);
```

Find the range and angle of a target located at (1000,2000,50) with respect to a local origin at (100,100,10). The local coordinate axes are `[1/sqrt(2) 1/sqrt(2) 0; 1/sqrt(2) -1/sqrt(2) 0; 0 0 1];`.

```
TargetLoc = [1e3;2e3;50];
refaxes =[1/sqrt(2) 1/sqrt(2) 0; 1/sqrt(2) -1/sqrt(2) 0; 0 0 1];
[tgtrng,tgtang] = rangeangle(TargetLoc,[100; 100; 10],refaxes);
```

See Also

`global2localcoord` | `local2globalcoord` | `azel2uv` | `azel2phitheta`

Related Examples

- “Global and Local Coordinate Systems”

Purpose

Range Doppler coupling

Syntax

```
dr = rdcoupling(fd,slope)
dr = rdcoupling(fd,slope,c)
```

Description

`dr = rdcoupling(fd,slope)` returns the range offset 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.

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

Range offset due to Doppler shift, returned as an array of real numbers. The dimensions of `dr` match the dimensions of `fd`.

Definitions

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)`.

Examples

Range of Target After Correcting for Doppler Shift

Calculate the true range of the target for an FMCW waveform that sweeps a band of 3 MHz in 2 ms. The dechirped target return has a beat frequency of 1 kHz. The processing of the target return also indicates a Doppler shift of 100 Hz.

```
slope = 30e6/2e-3;
fb = 1e3;
fd = 100;
r = beat2range(fb,slope) - rdcoupling(fd,slope);
```

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.

See Also

`beat2range` | `dechirp` | `range2beat` | `stretchfreq2rngphased.FMCWWaveform` **|** `phased.LinearFMWaveform` **|**

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

rocpfa

Purpose

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` pair arguments.

`rocpfa(...)` plots the ROC curves.

Input Arguments

Pfa

False-alarm probabilities in a row or column vector.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

MaxSNR

Maximum SNR to include in the ROC calculation.

Default: `20`

MinSNR

Minimum SNR to include in the ROC calculation.

Default: `0`

NumPoints

Number of SNR values to use when calculating the ROC curves. The actual values are equally spaced between `MinSNR` and `MaxSNR`.

Default: `101`

NumPulses

Number of pulses to integrate when calculating the ROC curves. A value of `1` indicates no pulse integration.

Default: `1`

SignalType

String that 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'`. The strings 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 $\texttt{erfc}^{-1}$ 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].

Default: `'NonfluctuatingCoherent'`

Output Arguments

Pd

Detection probabilities corresponding to the false-alarm probabilities. For each false-alarm probability in `Pfa`, `Pd` contains one column of detection probabilities.

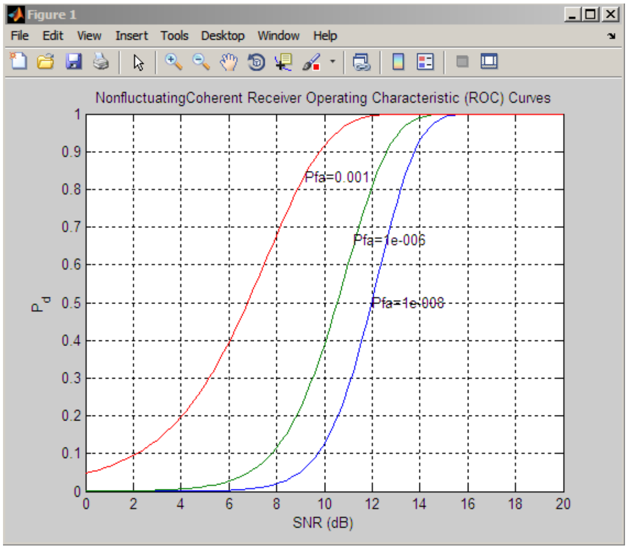
SNR

Signal-to-noise ratios in 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.

Examples

Plot ROC curves for false-alarm probabilities of 1e–8, 1e–6, and 1e–3, assuming coherent integration of a single pulse.

```
Pfa = [1e-8 1e-6 1e-3]; % false-alarm probabilities
rocpfa(Pfa,'SignalType','NonfluctuatingCoherent')
```



References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005, pp 298–336.

See Also

`npwgnthresh` | `rocsnr` | `shnidman`

Purpose

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` pair arguments.

`rocsnr(...)` plots the ROC curves.

Input Arguments

SNRdB

Signal-to-noise ratios in decibels, in a row or column vector.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

MaxPfa

Maximum false-alarm probability to include in the ROC calculation.

Default: `1`

MinPfa

Minimum false-alarm probability to include in the ROC calculation.

Default: `1e-10`

NumPoints

Number of false-alarm probabilities to use when calculating the ROC curves. The actual probability values are logarithmically equally spaced between `MinPfa` and `MaxPfa`.

Default: `101`

NumPulses

Number of pulses to integrate when calculating the ROC curves. A value of `1` indicates no pulse integration.

Default: `1`

SignalType

String that 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'`.

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 $\texttt{erfc}^{-1}$ 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].

Default: `'NonfluctuatingCoherent'`

Output Arguments

Pd

Detection probabilities corresponding to the false-alarm probabilities. For each SNR in `SNRdB`, `Pd` contains one column of detection probabilities.

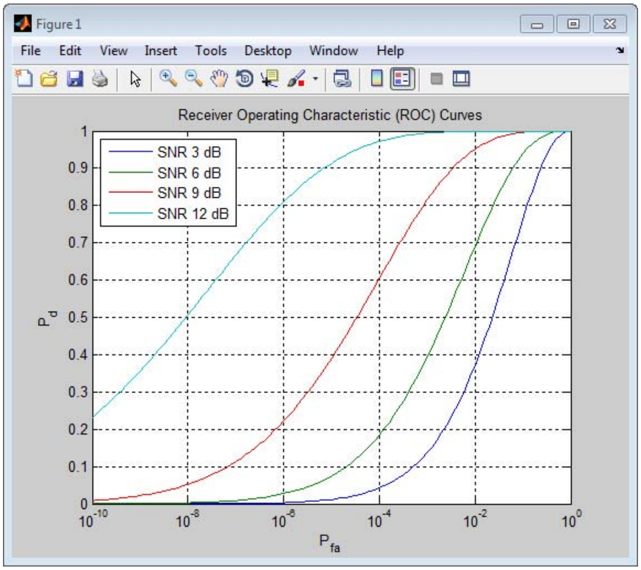
Pfa

False-alarm probabilities in 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.

Examples

Plot ROC curves for coherent integration of a single pulse.

```
SNRdB = [3 6 9 12]; % SNRs
[Pd,Pfa] = rocsnr(SNRdB,'SignalType','NonfluctuatingCoherent');
semilogx(Pfa,Pd);
grid on; xlabel('P_{fa}'); ylabel('P_d');
legend('SNR 3 dB','SNR 6 dB','SNR 9 dB','SNR 12 dB',...
    'location','northwest');
title('Receiver Operating Characteristic (ROC) Curves');
```



References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005, pp 298–336.

See Also

`npwgnthresh` | `rocpfa` | `shnidman`

Purpose

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,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,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.

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 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

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.

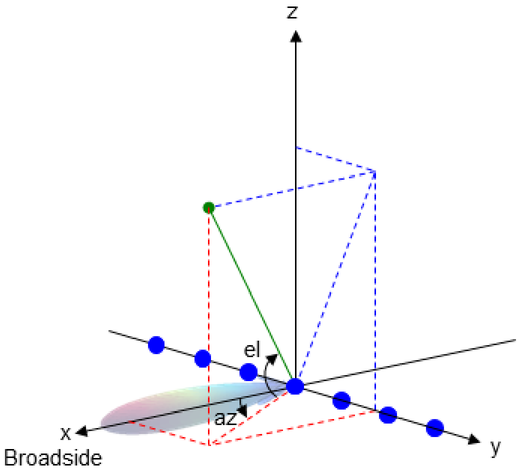
Definitions

Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive *x*-axis toward the positive *y*-axis, to the vector's orthogonal projection onto the *xy* plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the *xy* plane toward the positive *z*-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



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;
ha = 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 degrees, respectively. The noise is white across all array elements, and the SNR is 10 dB.

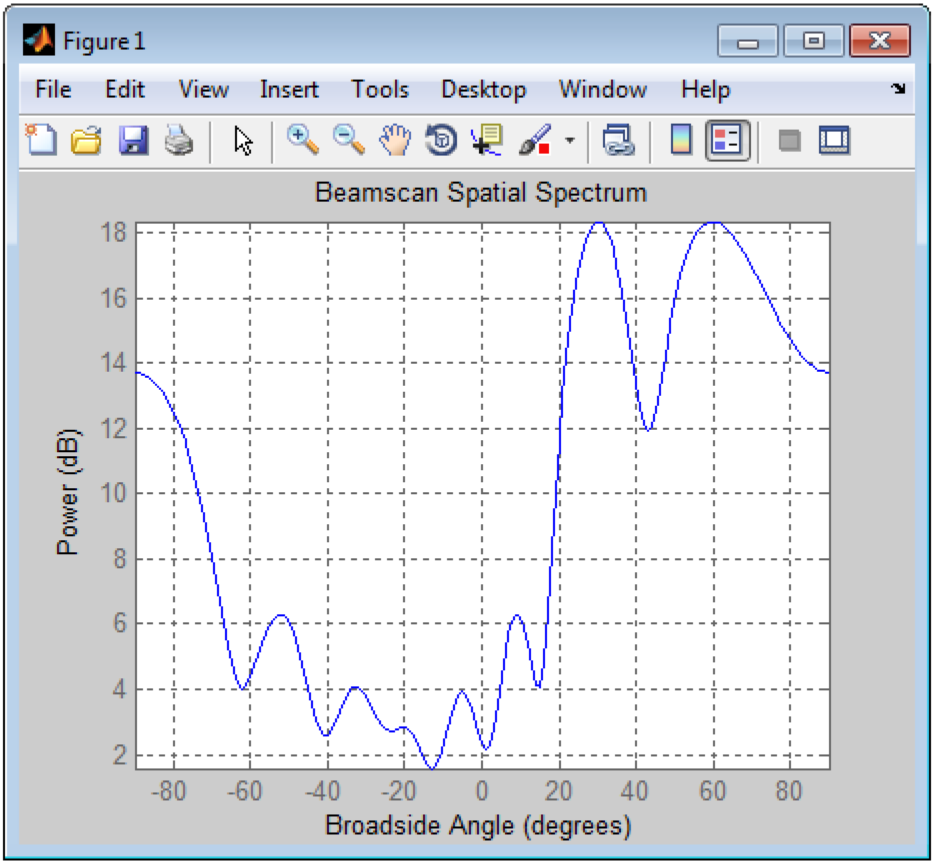
```
x = sensorsig(getElementPosition(ha)/lambda,...
   100,[30 60],db2pow(-10));
```

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 peaks at 30 and 60 degrees.

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.

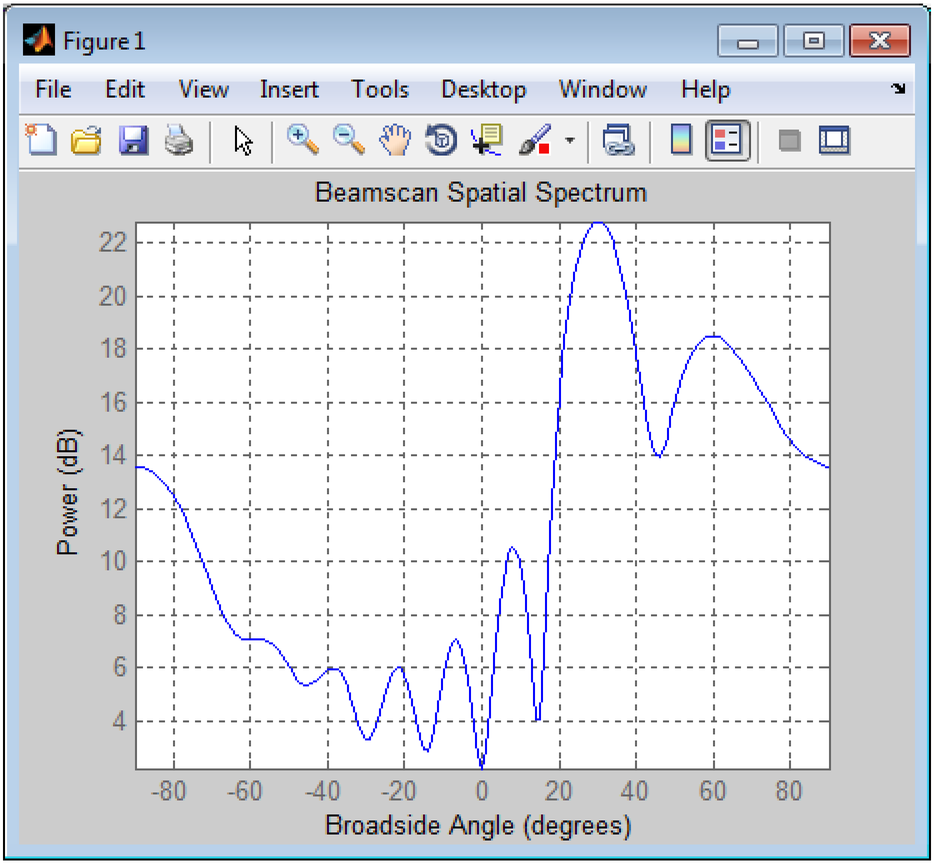
```
ang = [30 60];
scov = [3 1];
x = sensorsig(getElementPosition(ha)/lambda,...
   100,ang,db2pow(-10),scov);
```

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. A matrix named `scov` stores the signal covariance matrix.

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 degrees azimuth, respectively. The array that receives the signals is an 8-element uniform linear array whose elements are spaced half a 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's theoretical covariance, `rt`, with its sample covariance, `r`.

Create a 2-by-2 uniform rectangular array whose elements are spaced 1/4 of a 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's 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 degrees azimuth and 10 degrees elevation. A second incoming signal originates from 50 degrees azimuth and 0 degrees elevation. The signals have a power of 1 W and are not correlated with each other.

```
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)
abs(r)
```

```
ans =

    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


ans =

    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 Among Sensors

Simulate receiving a signal at a ULA, where the noise among different sensors is correlated.

Create a 4-element uniform linear array whose elements are spaced half a 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*th and *j*th array elements listed in `pos`.

```
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 degrees 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 =

   1.1000           -0.9027 - 0.4086i   0.6661 + 0.7458i  -0.3033 - 0.9529i
  -0.9027 + 0.4086i   1.1000           -0.9027 - 0.4086i   0.6661 + 0.7458i
   0.6661 - 0.7458i  -0.9027 + 0.4086i   1.1000           -0.9027 - 0.4086i
  -0.3033 + 0.9529i   0.6661 - 0.7458i  -0.9027 + 0.4086i   1.1000


r =

   1.1059           -0.8681 - 0.4116i   0.6550 + 0.7017i  -0.3151 - 0.9363i
  -0.8681 + 0.4116i   1.0037           -0.8458 - 0.3456i   0.6578 + 0.6750i
   0.6550 - 0.7017i  -0.8458 + 0.3456i   1.0260           -0.8775 - 0.3753i
  -0.3151 + 0.9363i   0.6578 - 0.6750i  -0.8775 + 0.3753i   1.0606
```

See Also

`phased.SteeringVector` |

Related Examples

- Direction of Arrival Estimation with Beamscan and MVDR

shnidman

Purpose

Required SNR using Shnidman's equation

Syntax

```
SNR = shnidman(Prob_Detect,Prob_FA)
SNR = shnidman(Prob_Detect,Prob_FA,N)
SNR = shnidman(Prob_Detect,Prob_FA,N, Swerling_Num)
```

Description

`SNR = shnidman(Prob_Detect,Prob_FA)` 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 of 0, a nonfluctuating target.

`SNR = shnidman(Prob_Detect,Prob_FA,N)` returns the required SNR for a nonfluctuating target based on the noncoherent integration of `N` pulses.

`SNR = shnidman(Prob_Detect,Prob_FA,N, Swerling_Num)` returns the required SNR for the Swerling case number `Swerling_Num`.

Definitions

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).

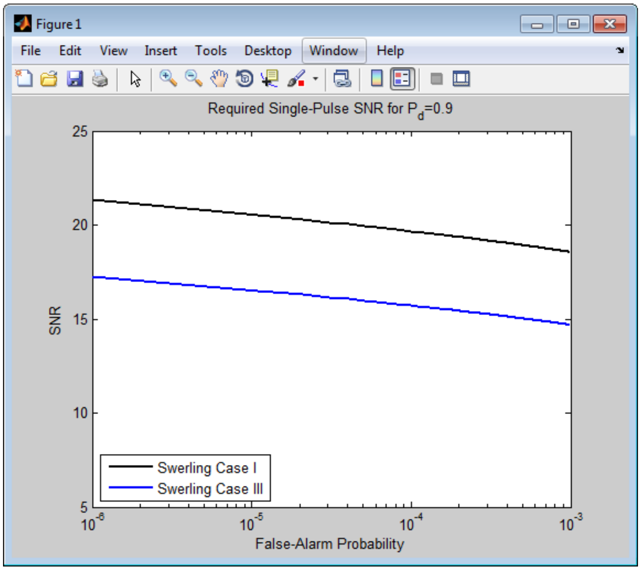
Swerling Case Number	Description
0 (alternatively designated as 5)	Nonfluctuating pulses.
I	Scan-to-scan decorrelation. Rayleigh/exponential PDF–A number of randomly distributed scatterers with no dominant scatterer.
II	Pulse-to-pulse decorrelation. Rayleigh/exponential PDF– A number of randomly distributed 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.

Examples

Find and compare the required single-pulse SNR for Swerling cases I and III.

```
Pfa = 1e-6:1e-5:.001; % False-alarm Probabilities
Pd = 0.9; % Probability of detection
SNR_Sw1 = zeros(1,length(Pfa)); % Preallocate space.
SNR_Sw3 = zeros(1,length(Pfa)); % Preallocate space.
for j=1:length(Pfa)
    % Swerling case I-No dominant scatterer
```

```
    SNR_Sw1(j) = shnidman(Pd,Pfa(j),1,1);
    % Swerling case III-Dominant scatterer
    SNR_Sw3(j) = shnidman(Pd,Pfa(j),1,3);
end
semilogx(Pfa,SNR_Sw1,'k','linewidth',2);
hold on;
semilogx(Pfa,SNR_Sw3,'b','linewidth',2);
axis([1e-6 1e-3 5 25]);
xlabel('False-Alarm Probability');
ylabel('SNR');
title('Required Single-Pulse SNR for P_d=0.9');
legend('Swerling Case I','Swerling Case III',...
    'Location','SouthWest');
```



Note that the presence of a dominant scatterer reduces the required SNR for the specified detection and false-alarm probabilities.

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005, p. 337.

See Also `albersheim`

speed2dop

Purpose

Convert speed to Doppler shift

Syntax

`Doppler_shift = speed2dop(radvel,lambda)`

Description

`Doppler_shift = speed2dop(radvel,lambda)` returns the one-way Doppler shift in hertz corresponding to the radial velocity, `radvel`, for the wavelength `lambda`.

Definitions

The following equation defines the Doppler shift in hertz based on the radial velocity of the source relative to the receiver and the carrier wavelength:

$$\Delta f = \frac{V_{s,r}}{\lambda}$$

where $V_{s,r}$ is the radial velocity of the source relative to the receiver in meters per second and λ is the wavelength in meters.

Examples

Calculate the Doppler shift in hertz for a given carrier wavelength and source speed.

```
radvel = 35.76; % 35.76 meters per second
f0= 24.15e9; % Frequency of 24.15 GHz
lambda = physconst('LightSpeed')/f0; % wavelength
Doppler_shift = speed2dop(radvel,lambda);
% Doppler shift of 2880.67 Hz
```

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`

Purpose

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`.

Input Arguments

FREQ

Frequency offset in hertz, specified as a scalar or vector.

SLOPE

Sweeping slope of the linear FM waveform, in hertz per second, specified as a nonzero scalar.

REFRNG

Reference range, in meters, specified as a scalar.

V

Propagation speed, in meters per second, specified as a positive scalar.

Default: Speed of light

Output Arguments

R

Range in meters. `R` has the same dimensions as `FREQ` .

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);
```

References

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

See Also

`phased.LinearFMWaveform` **|** `phased.StretchProcessor` **|** `ambgfun` | `beat2range` | `range2beat` | `rdcoupling`

Related Examples

- Range Estimation Using Stretch Processing

Concepts

- “Stretch Processing”

Purpose

Gamma value for different terrains

Syntax

```
G = surfacegamma(TerrainType)
G = surfacegamma(TerrainType,FREQ)
surfacegamma
```

Description

`G = surfacegamma(TerrainType)` returns the γ value for the specified terrain. The γ value is for an operating frequency of 10 GHz.

`G = surfacegamma(TerrainType,FREQ)` specifies the operating frequency of the system.

`surfacegamma` displays several terrain types and their corresponding γ values. These γ values are for an operating frequency of 10 GHz.

Input Arguments

TerrainType

String that describes type of terrain. Valid values are:

- `'sea state 3'`
- `'sea state 5'`
- `'woods'`
- `'metropolitan'`
- `'rugged mountain'`
- `'farmland'`
- `'wooded hill'`
- `'flatland'`

FREQ

Operating frequency of radar system in hertz. This value can be a scalar or vector.

Default: `10e9`

Output Arguments

G

Value of γ in decibels, for constant γ clutter model.

Definitions

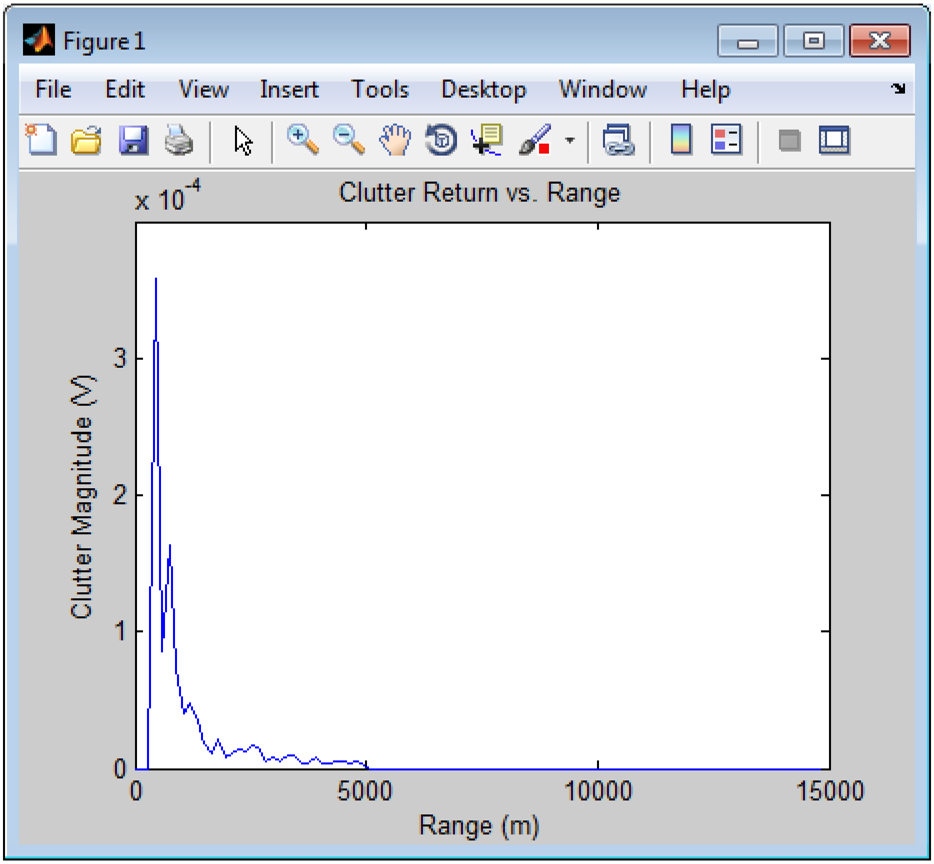
Gamma

A frequently used model for clutter simulation is the constant gamma model. This model uses a parameter, γ, to describe clutter characteristics of different types of terrain. Values of γ are derived from measurements.

Examples

Determine the γ value for a wooded area, and then simulate the clutter return from the area. Assume the radar system uses a single cosine pattern antenna element and an operating frequency of 300 MHz.

```
fc = 300e6;
g = surfacegamma('woods',fc);
hclutter = phased.ConstantGammaClutter('Gamma',g,...
    'Sensor',phased.CosineAntennaElement,...
    'OperatingFrequency',fc);
x = step(hclutter);
r = (0:numel(x)-1) / (2*hclutter.SampleRate) * ...
    hclutter.PropagationSpeed;
plot(r,abs(x));
xlabel('Range (m)'); ylabel('Clutter Magnitude (V)');
title('Clutter Return vs. Range');
```



Algorithms

The γ values for the terrain types `'sea state 3'`, `'sea state 5'`, `'woods'`, `'metropolitan'`, and `'rugged mountain'` are from [2].

The γ values for the terrain types `'farmland'`, `'wooded hill'`, and `'flatland'` are from [3].

Measurements provide values of γ for a system operating at 10 GHz. The γ value for a system operating at frequency *f* is:

$$\gamma = \gamma_0 + 5\log\left(\frac{f}{f_0}\right)$$

where γ_0 is the value at frequency f_0 = 10 GHz.

References

[1] Barton, David. "Land Clutter Models for Radar Design and Analysis," *Proceedings of the IEEE*. Vol. 73, Number 2, February, 1985, pp. 198–204.

[2] Long, Maurice W. *Radar Reflectivity of Land and Sea*, 3rd Ed. Boston: Artech House, 2001.

[3] Nathanson, Fred E., J. Patrick Reilly, and Marvin N. Cohen. *Radar Design Principles*, 2nd Ed. Mendham, NJ: SciTech Publishing, 1999.

See Also

`grazingang` | `horizonrange` `phased.ConstantGammaClutter` |

Purpose

Surface clutter radar cross section (RCS)

Syntax

```
RCS = surfclutterrcs(NRCS,R,az,el,graz,tau)
RCS = surfclutterrcs(NRCS,R,az,el,graz,tau,c)
```

Description

`RCS = surfclutterrcs(NRCS,R,az,el,graz,tau)` returns the radar cross section (RCS) of a clutter patch that is of range `R` meters away from the radar system. `az` and `el` are the radar system azimuth and elevation beamwidths, respectively, corresponding to the clutter patch. `graz` is the grazing angle of the clutter patch relative to the radar. `tau` is the pulse width of the transmitted signal. The calculation automatically determines whether the surface clutter area is beam limited or pulse limited, based on the values of the input arguments.

`RCS = surfclutterrcs(NRCS,R,az,el,graz,tau,c)` specifies the propagation speed in meters per second.

Tips

- You can calculate the clutter-to-noise ratio using the output of this function as the `RCS` input argument value in `radareqsnr`.

Input Arguments

NRCS

Normalized radar cross section of clutter patch in units of square meters/square meters.

R

Range of clutter patch from radar system, in meters.

az

Azimuth beamwidth of radar system corresponding to clutter patch, in degrees.

el

Elevation beamwidth of radar system corresponding to clutter patch, in degrees.

graz

Grazing angle of clutter patch relative to radar system, in degrees.

tau

Pulse width of transmitted signal, in seconds.

c

Propagation speed, in meters per second.

Default: Speed of light

Output Arguments

RCS

Radar cross section of clutter patch.

Examples

Calculate the RCS of a clutter patch and estimate the clutter-to-noise ratio at the receiver. Assume that the patch has an NRCS of 1 m^2/m^2 and is 1000 m away from the radar system. The azimuth and elevation beamwidths are 1 degree and 3 degrees, respectively. The grazing angle is 10 degrees. The pulse width is 10 µs. The radar is operated at a wavelength of 1 cm with a peak power of 5 kw.

```
nrcs = 1; rng = 1000;
az = 1; el = 3; graz = 10;
tau = 10e-6; lambda = 0.01; ppow = 5000;
rcs = surfclutterrcs(nrcs,rng,az,el,graz,tau);
cnr = radareqsnr(lambda,rng,ppow,tau,'rcs',rcs);
```

Algorithms

See [1].

References

[1] Richards, M. A. *Fundamentals of Radar Signal Processing*. New York: McGraw-Hill, 2005, pp. 57–63.

See Also

`grazingang` | `surfacegamma` | `radareqsnr` | `uv2azel` | `phitheta2azel`

Purpose

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`. The reference temperature is 290 K.

`STEMP = systemp(NF,REFTEMP)` specifies the reference temperature.

Input Arguments

NF

Noise figure in decibels. The noise figure is the ratio of the actual output noise power in a receiver to the noise power output of an ideal receiver.

REFTEMP

Reference temperature in kelvin, 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.

Default: `290`

Output Arguments

STEMP

Effective system-noise temperature in kelvin. The effective system-noise temperature is `REFTEMP*10^(NF/10)`.

Examples

Calculate the system-noise temperature of a receiver with a 300 K reference temperature and a 5 dB noise figure.

```
stemp = systemp(5,300);
```

References

[1] Skolnik, M. *Introduction to Radar Systems*. New York: McGraw-Hill, 1980.

See Also `noisepow` `phased.ReceiverPreamp` |

Purpose

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.

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`.

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);
```

References

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

See Also

`range2time` | `range2bw` `phased.FMCWWaveform` |

Purpose Uniform grid

Syntax

```
Grid = unigrid(StartValue,Step,EndValue)
Grid = unigrid(StartValue,Step,EndValue,IntervalType)
```

Description

`Grid = unigrid(StartValue,Step,EndValue)` returns a uniformly sampled grid from the closed interval `[StartValue,EndValue]`, starting from `StartValue`. `Step` specifies the step size. This syntax is the same as calling `StartValue:Step:EndValue`.

`Grid = unigrid(StartValue,Step,EndValue,IntervalType)` specifies whether the interval is closed, or semi-open. Valid values of `IntervalType` are `'[]'` (default), and `'[)'`. Specifying a closed interval does not always cause `Grid` to contain the value `EndValue`. The inclusion of `EndValue` in a closed interval also depends on the step size `Step`.

Examples

Create a uniform closed interval with a positive step.

```
Grid = unigrid(0,0.1,1);
% Note that Grid(1)=0 and Grid(end)=1
```

Create semi-open interval.

```
Grid = unigrid(0,0.1,1,'[)');
% Grid(1)=0 and Grid(end)=0.9
```

See Also

`linspace` | `val2ind`

uv2azel

Purpose

Convert u/v coordinates to azimuth/elevation angles

Syntax

`AzEl = uv2azel(UV)`

Description

`AzEl = uv2azel(UV)` converts the *u*/*v* space coordinates to their corresponding azimuth/elevation angle pairs.

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`.

Definitions

U/V Space

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, as follows:

$$u = \sin(\theta)\cos(\varphi)$$

$$v = \sin(\theta)\sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

The values of *u* and *v* satisfy these inequalities:

$$-1 \leq u \leq 1$$

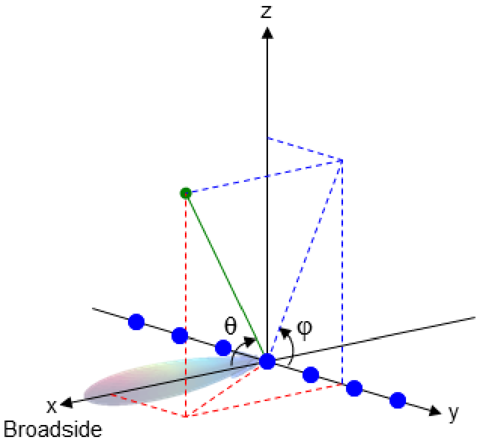
$$-1 \leq v \leq 1$$

$$u^2 + v^2 \leq 1$$

Phi Angle, Theta Angle

The φ angle is the angle from the positive y-axis toward the positive z-axis, to the vector's orthogonal projection onto the yz plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the x-axis toward the yz plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



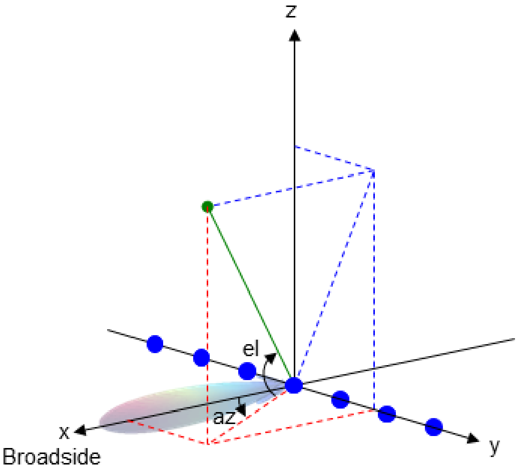
Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive x-axis toward the positive y-axis, to the vector's orthogonal projection onto the xy plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the xy

plane toward the positive z-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive x-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of U/V Coordinates

Find the corresponding azimuth/elevation representation for $u = 0.5$ and $v = 0$.

```
AzEl = uv2azel([0.5; 0]);
```

See Also `azel2uv`

Concepts

- “Spherical Coordinates”

uv2azelpat

Purpose

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,el] = uv2azelpat( ___ )
```

Description

`pat_azel = uv2azelpat(pat_uv,u,v)` expresses the antenna radiation pattern `pat_azel` in azimuth/elevation angle coordinates instead of u/v space 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,el] = 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.

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 form

M-by-L matrix

Antenna radiation pattern in azimuth/elevation form, returned as an M-by-L matrix. `pat_azel` samples the 3-D magnitude pattern in decibels, in terms of azimuth and elevation angles. L is the length of the `az` vector, and M is the length of the `el` vector.

az - Azimuth angles

vector of length L

Azimuth angles at which `pat_azel` samples the pattern, returned as a vector of length L. Angles are expressed in degrees.

el - Elevation angles

vector of length M

Elevation angles at which `pat_azel` samples the pattern, returned as a vector of length M. Angles are expressed in degrees.

Definitions

U/V Space

The u/v coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, as follows:

$$u = \sin(\theta) \cos(\varphi)$$

$$v = \sin(\theta) \sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

The values of u and v satisfy these inequalities:

$$-1 \leq u \leq 1$$

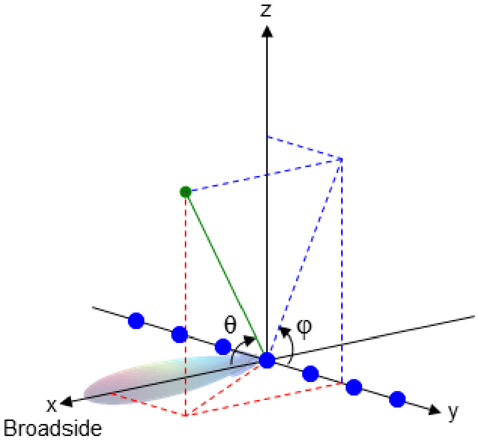
$$-1 \leq v \leq 1$$

$$u^2 + v^2 \leq 1$$

Phi Angle, Theta Angle

The φ angle is the angle from the positive y-axis toward the positive z-axis, to the vector's orthogonal projection onto the yz plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the *x*-axis toward the *yz* plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.

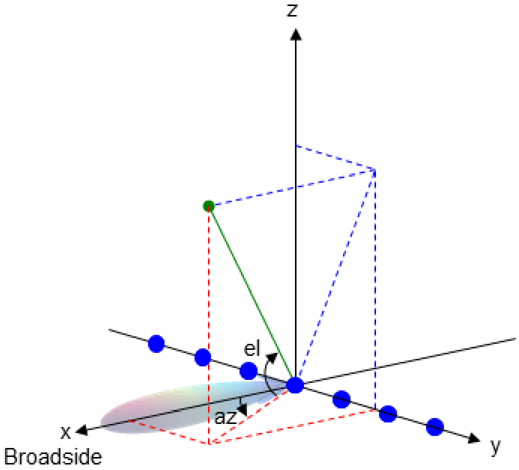


Azimuth Angle, Elevation Angle

The *azimuth angle* is the angle from the positive *x*-axis toward the positive *y*-axis, to the vector's orthogonal projection onto the *xy* plane. The azimuth angle is between –180 and 180 degrees. The *elevation angle* is the angle from the vector's orthogonal projection onto the *xy* plane toward the positive *z*-axis, to the vector. The elevation angle is between –90 and 90 degrees. These definitions assume the boresight direction is the positive *x*-axis.

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 that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of Radiation Pattern

Convert a radiation pattern to azimuth/elevation form, with the angles spaced 1 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 azimuth/elevation space.

```
pat_azel = uv2azelpat(pat_uv,u,v);
```

Plot of Converted Radiation Pattern

Convert a radiation pattern to azimuth/elevation form, with the angles spaced 1 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.

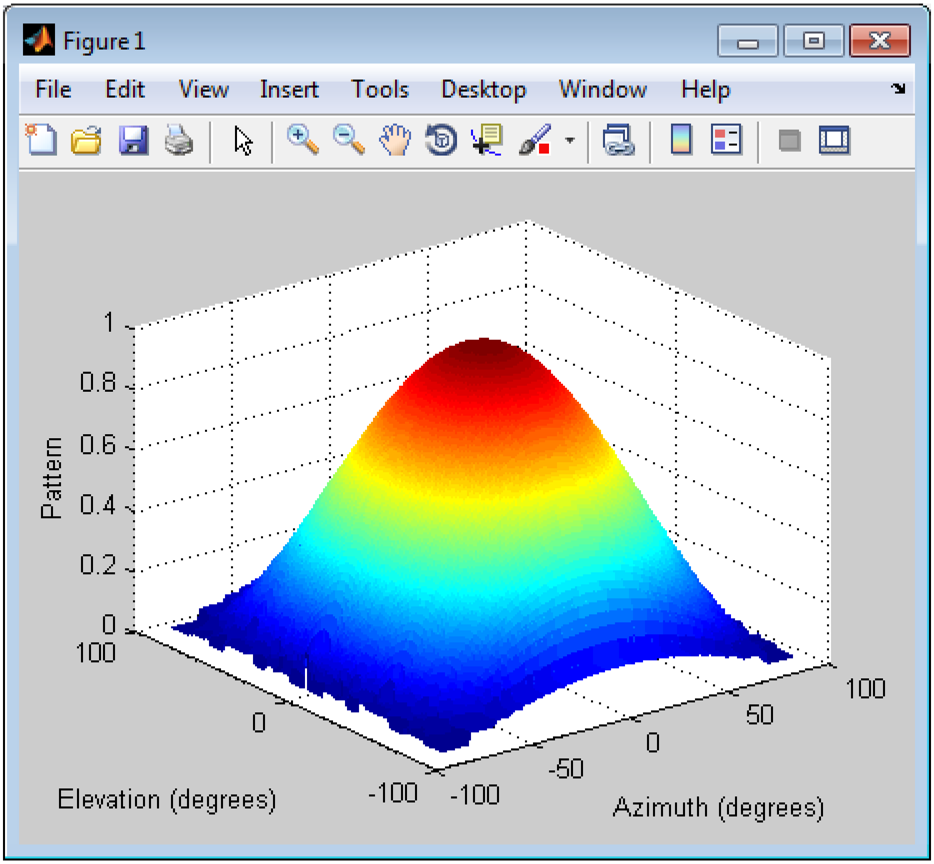
```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Convert the pattern to azimuth/elevation space. Store the azimuth and elevation angles to use them for plotting.

```
[pat_azel,az,el] = uv2azelpat(pat_uv,u,v);
```

Plot the result.

```
H = surf(az,el,pat_azel);
set(H,'LineStyle','none')
xlabel('Azimuth (degrees)');
ylabel('Elevation (degrees)');
zlabel('Pattern');
```



Conversion of Radiation Pattern Using Specific Azimuth/Elevation Values

Convert a radiation pattern to azimuth/elevation form, with the angles spaced 5 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;
```

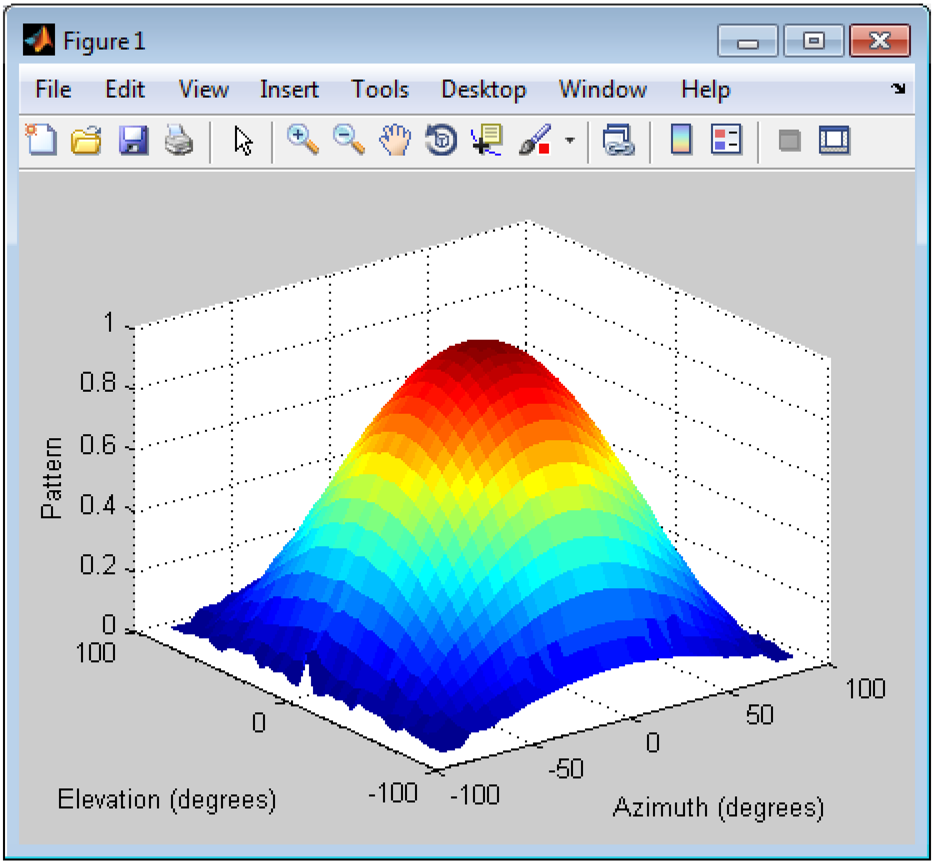
```
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Define the set of 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 result.

```
H = surf(az,el,pat_azel);
set(H,'LineStyle','none')
xlabel('Azimuth (degrees)');
ylabel('Elevation (degrees)');
zlabel('Pattern');
```



See Also `phased.CustomAntennaElement` | `uv2azel` | `azel2uv` | `azel2uvpat`

Concepts

- "Spherical Coordinates"

Purpose

Convert u/v coordinates to phi/theta angles

Syntax

`PhiTheta = uv2phitheta(UV)`

Description

`PhiTheta = uv2phitheta(UV)` converts the *u*/*v* space coordinates to their corresponding phi/theta angle pairs.

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`.

Definitions

U/V Space

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, as follows:

$$u = \sin(\theta)\cos(\varphi)$$

$$v = \sin(\theta)\sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

The values of u and v satisfy these inequalities:

$-1 \le u \le 1$

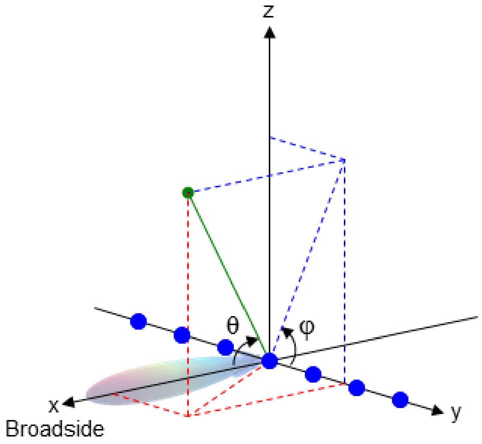
$-1 \le v \le 1$

$u^2 + v^2 \le 1$

Phi Angle, Theta Angle

The φ angle is the angle from the positive *y*-axis toward the positive *z*-axis, to the vector's orthogonal projection onto the *yz* plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the *x*-axis toward the *yz* plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of U/V Coordinates

Find the corresponding φ/θ representation for $u = 0.5$ and $v = 0$.

```
PhiTheta = uv2phitheta([0.5; 0]);
```

See Also `phitheta2uv`

Concepts

- "Spherical Coordinates"

Purpose

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,theta] = uv2phithetapat( ___ )
```

Description

`pat_phitheta = uv2phithetapat(pat_uv,u,v)` expresses the antenna radiation pattern `pat_phitheta` in φ/θ angle coordinates instead of u/v space 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,theta] = 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.

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 form

M-by-L matrix

Antenna radiation pattern in phi/theta form, returned as an M-by-L matrix. `pat_phitheta` samples the 3-D magnitude pattern in decibels, in terms of φ and θ angles. L is the length of the `phi` vector, and M is the length of the `theta` vector.

phi - Phi angles

vector of length L

Phi angles at which `pat_phitheta` samples the pattern, returned as a vector of length L. Angles are expressed in degrees.

theta - Theta angles

vector of length M

Theta angles at which `pat_phitheta` samples the pattern, returned as a vector of length M. Angles are expressed in degrees.

Definitions

U/V Space

The u/v coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, as follows:

$$u = \sin(\theta)\cos(\varphi)$$

$$v = \sin(\theta)\sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

The values of u and v satisfy these inequalities:

$$-1 \leq u \leq 1$$

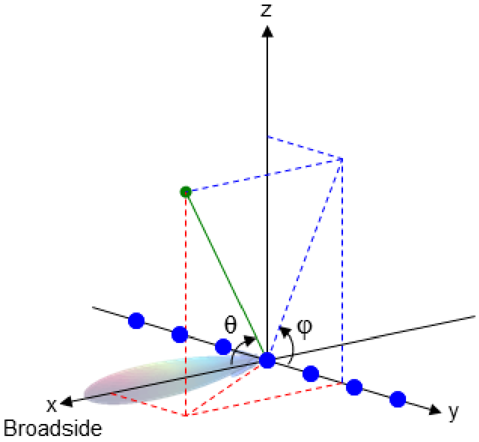
$$-1 \leq v \leq 1$$

$$u^2 + v^2 \leq 1$$

Phi Angle, Theta Angle

The φ angle is the angle from the positive y-axis toward the positive z-axis, to the vector's orthogonal projection onto the yz plane. The φ angle is between 0 and 360 degrees. The θ angle is the angle from the x-axis toward the yz plane, to the vector itself. The θ angle is between 0 and 180 degrees.

The figure illustrates φ and θ for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



Examples

Conversion of Radiation Pattern

Convert a radiation pattern to φ/θ form, with the angles spaced 1 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 φ/θ space.

```
[pat_phitheta,phi,theta] = uv2phithetapat(pat_uv,u,v);
```

Plot of Converted Radiation Pattern

Convert a radiation pattern to φ/θ form, with the angles spaced 1 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.

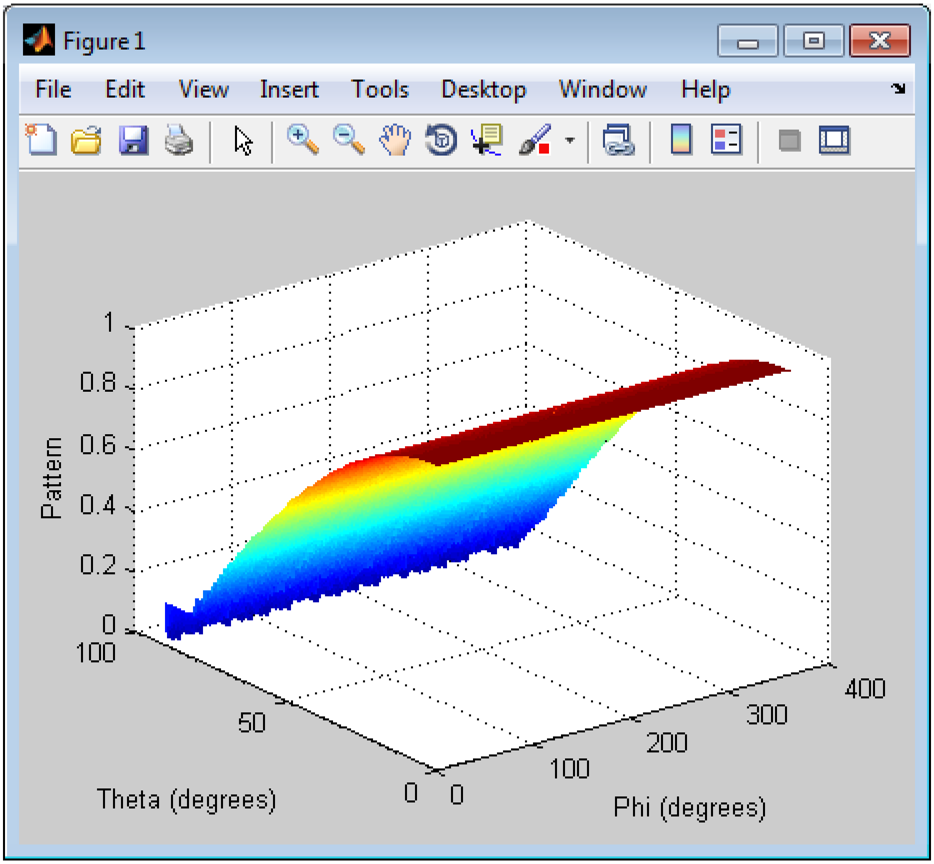
```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Convert the pattern to φ/θ space. Store the φ and θ angles to use them for plotting.

```
pat_phitheta = uv2phithetapat(pat_uv,u,v);
```

Plot the result.

```
H = surf(phi,theta,pat_phitheta);
set(H,'LineStyle','none')
xlabel('Phi (degrees)');
ylabel('Theta (degrees)');
zlabel('Pattern');
```



Conversion of Radiation Pattern Using Specific Phi/Theta Values

Convert a radiation pattern to φ/θ form, with the angles spaced 5 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;
```

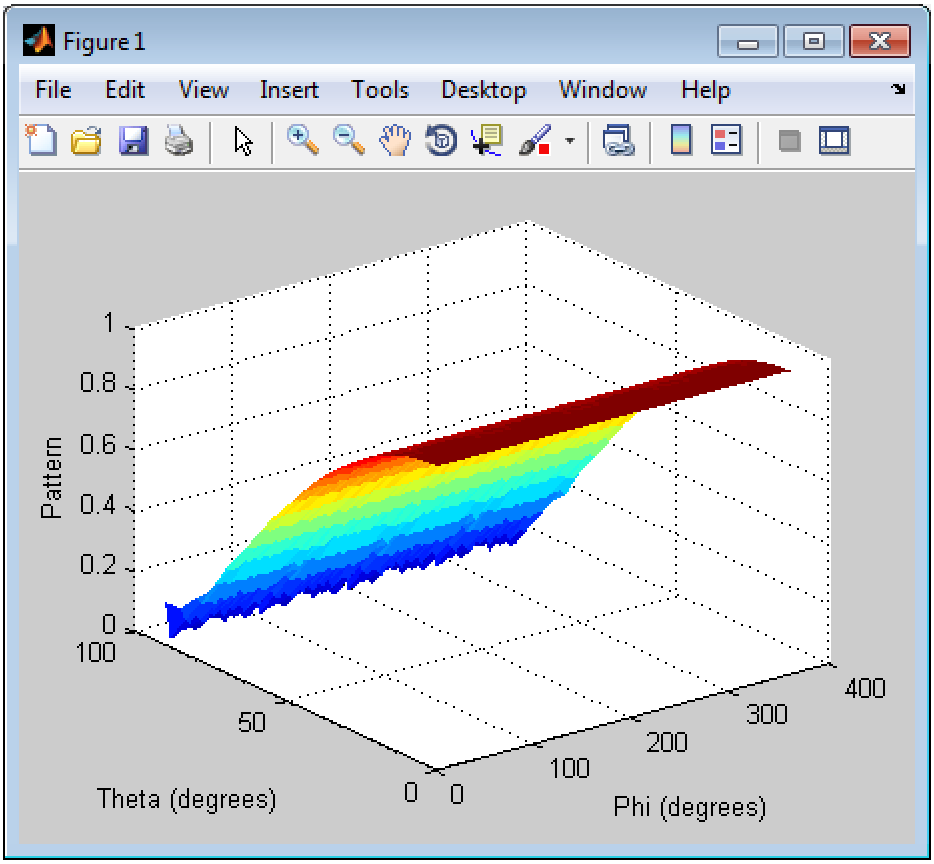
```
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

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);
set(H,'LineStyle','none')
xlabel('Phi (degrees)');
ylabel('Theta (degrees)');
zlabel('Pattern');
```



See Also

`phased.CustomAntennaElement` | `uv2phitheta` | `phitheta2uv` | `phitheta2uvpat`

Concepts

- "Spherical Coordinates"

val2ind

Purpose

Uniform grid index

Syntax

```
Ind = val2ind(Value,Delta)
Ind = val2ind(Value,Delta,GridStartValue)
```

Description

`Ind = val2ind(Value,Delta)` returns the index of the value `Value` in a uniform grid with a spacing between elements of `Delta`. The first element of the uniform grid is zero. If `Value` does not correspond exactly to an element of the grid, the next element is returned. If `Value` is a row vector, `Ind` is a row vector of the same size.

`Ind = val2ind(Value,Delta,GridStartValue)` specifies the starting value of the uniform grid as `GridStartValue`.

Examples

Find index for 0.001 in uniform grid with 1 MHz sampling rate.

```
Fs = 1e6;
Ind = val2ind(0.001,1/Fs);
% Ind is 1001 because the 1st grid element is zero
```

Find indices for vector with 1 kHz sampling rate.

```
Fs = 1e3;
% Construct row vector of values
Values =[0.0095 0.0125 0.0225];
% Values not divisible by 1/Fs
% with nonzero remainder
Ind = val2ind(Values,1/Fs);
% Returns Ind =[11 14 24]
```