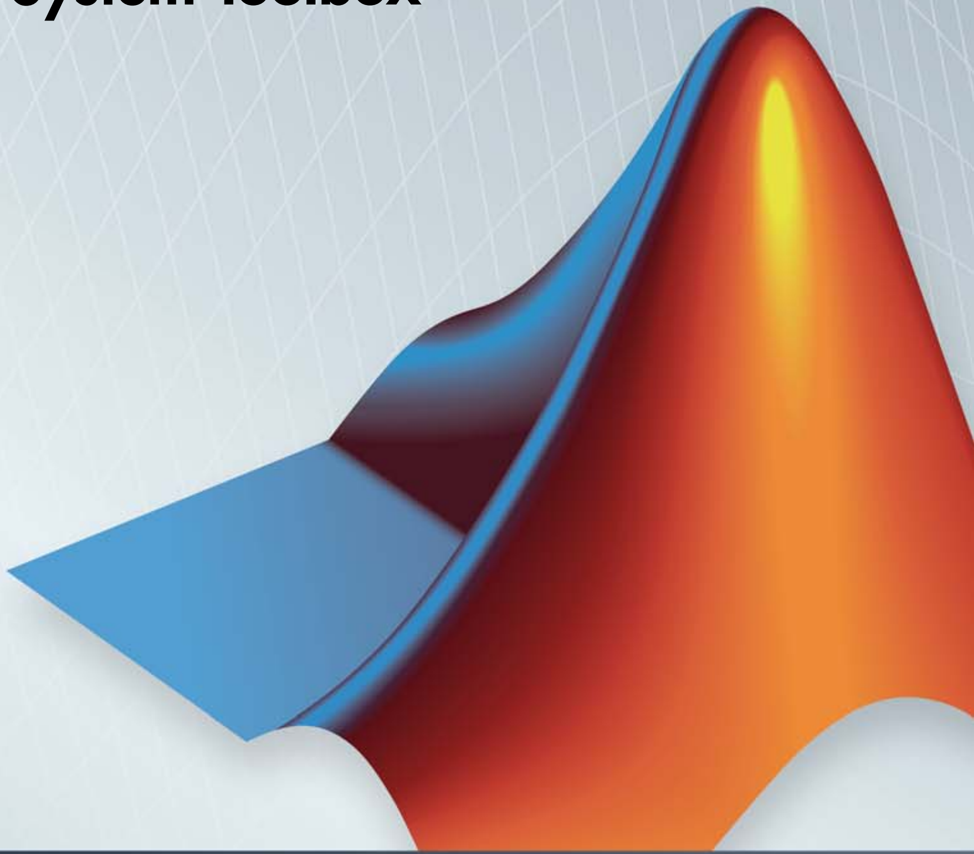
Phased Array System Toolbox™

Reference

R2014**a**



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 2011–2014 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)
March 2013	Online only	Revised for Version 2.0 (R2013a)
September 2013	Online only	Revised for Version 2.1 (R2013b)
March 2014	Online only	Revised for Version 2.2 (R2014a)

Contents

1 Alphabetical List

2 Functions-Alphabetical List

1

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. Type 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

cloneImpl	Copy System object
getDiscreteStateImpl	Discrete state property values
getNumInputsImpl	Number of input arguments passed to step and setup methods
getNumOutputsImpl	Number of outputs returned by method
infoImpl	Information about System object™
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 Basic System Object

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
```

Use this object by creating an instance of `AddOne`, providing an input, and using 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
```

See Also `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.cloneImpl

Purpose

Copy System object

Syntax

`cloneImpl(obj)`

Description

`cloneImpl(obj)` copies a System object by using the `saveObjectImpl` and `loadObjectImpl` methods. The default `cloneImpl` copies an object and its current state but does not copy any private or protected properties. If the object you clone is locked and you use the default `cloneImpl`, the new object will also be locked. If you define your own `cloneImpl` and the associated `saveObjectImpl` and `loadObjectImpl`, you can specify whether to clone the object's state and whether to clone the object's private and protected properties.

`cloneImpl` is called by the `clone` method.

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

You cannot modify any properties in this method.

Input Arguments

obj

System object handle of object to clone.

Examples

Clone a System Object

Use the `cloneImpl` method in your class definition file to copy a System object

```
methods (Access=protected)
   function obj2 = cloneImpl(obj1)
      s = saveObject (obj1);
      obj2 = loadObject(s);
   end
end
```

See Also

`saveObjectImpl` | `saveObjectImpl`

How To

- "Clone System Object"

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.

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

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

You cannot modify any properties in this method.

Input Arguments

obj

System object handle

Output Arguments

s

Struct of state values.

Examples

Get Discrete State Values

Use the `getDiscreteStateImpl` method in your class definition file to get the discrete states of the object.

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

See Also

`setupImpl`

How To

- “Define Property Attributes”

matlab.System.getNumInputsImpl

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.

If your `step` method has a variable number of inputs (uses `varargin`), you should implement the `getNumInputsImpl` method in your class definition file. If the number of inputs expected by the `step` method is fixed (does not use `varargin`), the default `getNumInputsImpl` determines the required number of inputs directly from the `step` method. In this case, you do not need to include `getNumInputsImpl` 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.

You cannot modify any properties in 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

Set Number of Inputs

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
```

Set Number of Inputs to Zero

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

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

See Also

`setupImpl` | `stepImpl` | `getNumOutputsImpl`

How To

- “Change Number of Step Inputs or Outputs”

matlab.System.getNumOutputsImpl

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.

You cannot modify any properties in 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

Set Number of Outputs

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
```

Set Number of Outputs to Zero

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

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

See Also

`stepImpl` | `getNumInputsImpl` | `setupImpl`

How To

- “Change Number of Step Inputs or Outputs”

matlab.System.infoImpl

Purpose

Information about System object

Syntax

`s = infoImpl(obj,varargin)`

Description

`s = infoImpl(obj,varargin)` lets you set up information to return about the current configuration of a System object `obj`. This information is returned in a struct from the `info` method. The `varargin` argument is optional. The default `infoImpl` method, which is used if you do not include `infoImpl` in your class definition file, returns an empty struct.

`infoImpl` is called by the `info` method.

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

Input Arguments

obj

System object handle

varargin

Allows variable number of inputs

Examples

Define info for System object

Define the `infoImpl` method to return current count information for `info(obj)`.

```
methods (Access=protected)
   function s = infoImpl(obj)
      s = struct('Count',obj.pCount);
   end
end
```

How To

- "Define System Object Information"

matlab.System.isInactivePropertyImpl

Purpose

Active or inactive flag for properties

Syntax

`flag = isInactivePropertyImpl(obj,prop)`

Description

`flag = isInactivePropertyImpl(obj,prop)` specifies whether a public, non-state 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

Public, non-state property name

Output Arguments

flag

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

Examples

Set Inactive Property

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
```

See Also `setProperties`

How To

- “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.

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

Input Arguments

obj

System object handle

Examples

Load System Object

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
```

See Also

`saveObjectImpl`

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.

You cannot modify any tunable properties in this method if its System object will be used in the Simulink® MATLAB System block.

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

Specify Action When Tunable Property Changes

Use `processTunedPropertiesIempl` 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
```

See Also `validatePropertiesImpl` | `setProperties`

How To

- "Validate Property and Input Values"
- "Define Property Attributes"

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

Close a File and Release Its Resources

Use the `releaseImpl` method to close a file.

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

How To

- “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. This is useful for initialization at the start of simulation.

`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.

Do not use `resetImpl` to initialize or reset properties. For properties, use the `setupImpl` method.

You cannot modify any tunable properties in this method if its System object will be used in the Simulink MATLAB System block.

Input Arguments

obj

System object handle

Examples

Reset Property Value

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

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

See Also

`releaseImpl`

How To

- "Reset Algorithm State"

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 and properties with the `DiscreteState` attribute 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.

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

Input Arguments

obj

System object handle

Examples

Define Property and State Values to Save

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 properties. 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
```

See Also

`loadObjectImpl`

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,...
   'ValueOnlyPropName1','ValueOnlyPropName2',...,
   'ValueOnlyPropNamem')
```

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,...` `'ValueOnlyPropName1','ValueOnlyPropName2',...,` `'ValueOnlyPropNamem')` 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 objectSystem object handle

numargs

Number of inputs passed in by the object constructor

name1,name2,...

Name of property

value1,value2,...

Value of the property

arg1,arg2,...

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

ValueOnlyPropName1,ValueOnlyPropName2,...

Name of the value-only property

Examples

Setup Value-Only Inputs

Set up an object so users can specify value-only inputs for `VProp1`, `VProp2`, and other property values via name-value pairs when constructing the object. In this example, `VProp1` and `VProp2` are the names of value-only properties.

```
methods
   function obj = MyFile(varargin)
      setProperties(obj,nargin,varargin{:},'VProp1','VProp2');
   end
end
```

How To

- “Set Property Values at Construction Time”

Purpose

Initialize System object

Syntax

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

Description

`setupImpl(obj)` sets up a System object and implements one-time tasks that do not depend on any inputs to the `stepImpl` method for this 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. You typically use `setupImpl` to set private properties so they do not need to be calculated each time `stepImpl` method is called.

`setupImpl(obj,input1,input2,...)` sets up a System object using one or more of the `stepImpl` input specifications. The number and order of inputs must match the number and order of inputs defined in the `stepImpl` method. You pass the inputs into `setupImpl` to use the specifications, such as size and datatypes in the one-time calculations. You do not use the `setupImpl` method to set up input values.

`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 can omit this method from your class definition file if your System object does not require any setup tasks.

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

Do not use `setupImpl` to initialize or reset states. For states, use the `resetImpl` method.

You cannot modify any tunable properties in this method if its System object will be used in the Simulink MATLAB System block.

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

input1,input2,...

Inputs to the `stepImpl` method

Examples

Setup a File for Writing

This example shows how to open a file for writing using the `setupImpl` method in your class definition file.

```
methods (Access=protected)
   function setupImpl(obj)
      obj.pFileID = fopen(obj.Filename,'wb');
      if obj.pFileID < 0
         error('Opening the file failed');
       end
    end
end
```

Check input size

This examples shows how to use `setupImpl` to check that the size of a `stepImpl` method input matches the size of a state property.

```
properties (Access = private)
     myState = [1 2];
  end

  methods (Access = protected)
     function setupImpl(obj,u)
        if any(size(obj.myState) ~= size(u))
           error('Size of "myState" does not match size of input "u"');
        end
```

```
      end

      function y = stepImpl(obj,u)
         y = obj.myState;
         obj.myState = u;
      end
   end
end
```

See Also `validatePropertiesImpl` | `validateInputsImpl` | `setProperties`

How To

- “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

input1,input2,...

Inputs to the `step` method

Output Arguments

output

Output returned from the `step` method.

Examples

Specify System Object Algorithm

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
  end
```

See Also

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

How To

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

matlab.System.validateInputsImpl

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.

You cannot modify any properties in this method. Use the `processTunedPropertiesImpl` method or `setupImpl` method to modify properties.

Input Arguments

obj

System object handle

input1,input2,...

Inputs to the `setup` method

Examples

Validate Input Type

Validate that the input is numeric.

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

See Also `validatePropertiesImpl` | `setupImpl`

How To

- "Validate Property and Input Values"

matlab.System.validatePropertiesImpl

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.

You cannot modify any properties in this method. Use the `processTunedPropertiesImpl` method or `setupImpl` method to modify properties.

Input Arguments

obj

System object handle

Examples

Validate a Property

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
```

See Also

`processTunedPropertiesImpl` | `setupImpl` | `validateInputsImpl`

How To

- “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. Type 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

See Also `matlab.System`

Tutorials

- "Define Finite Source Objects"

How To

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

matlab.system.mixin.FiniteSource.isDoneImpl

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.

Note You must set `Access=protected` for this 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

Check for End-of-Data

Set up the `isDoneImpl` method in your class definition file so the `isDone` method checks whether the object has completed eight iterations.

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

See Also

`matlab.system.mixin.FiniteSource`

How To

- “Define Finite Source Objects”

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 String Property Values

Set the string property, `Flavor`, and the `StringSet` property, `FlavorSet` in your class definition file.

```
properties
  Flavor='Chocolate';
end

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

```
end
```

See Also `matlab.System`

How To

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

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 1-41.
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 1-41 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

Process radar data cube using ADPCA processor.

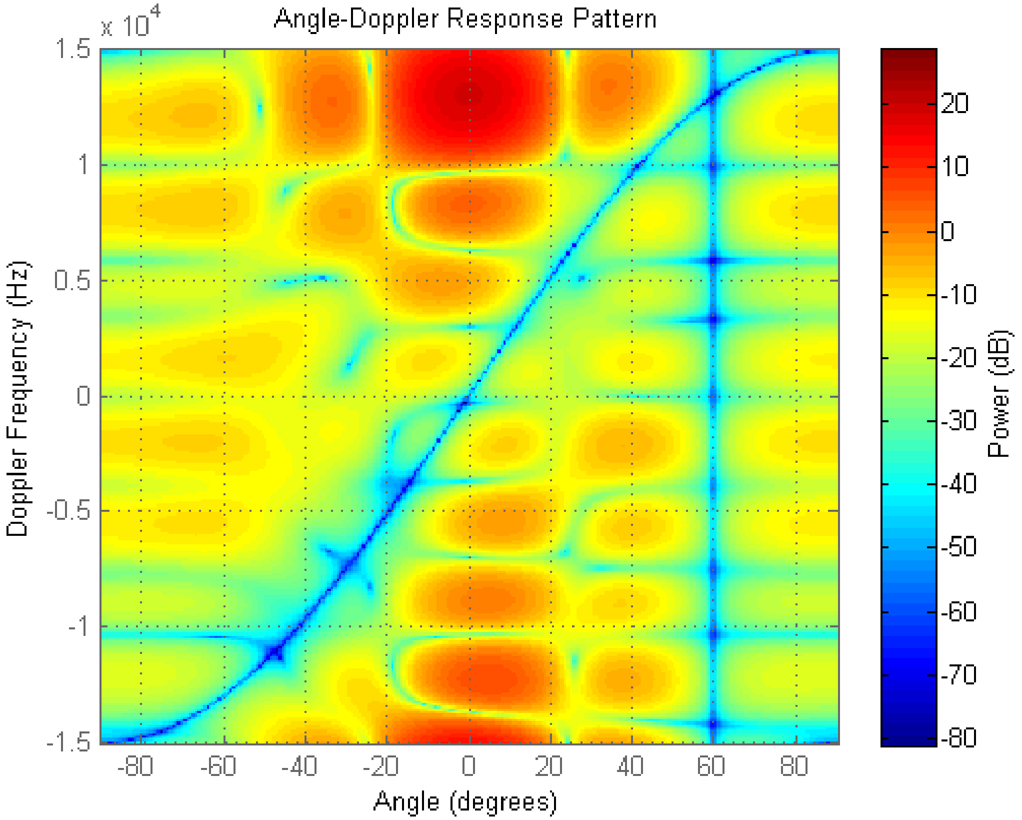
Process a radar data cube using an ADPCA processor. The weights are calculated for the 71st cell of the data cube. Set the look direction to [0;0] degrees and the Doppler shift to 12980 Hz.

Load radar data file and compute weights

```
load STAPExampleData;
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);
```

Create AnglerDoppler System object and plot response

```
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`.

phased.ADPCACanceller.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.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.

---

phased.ADPCACanceller.step

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 1-56.

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

Sensor array

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

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

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

Calculate Angle-Doppler response

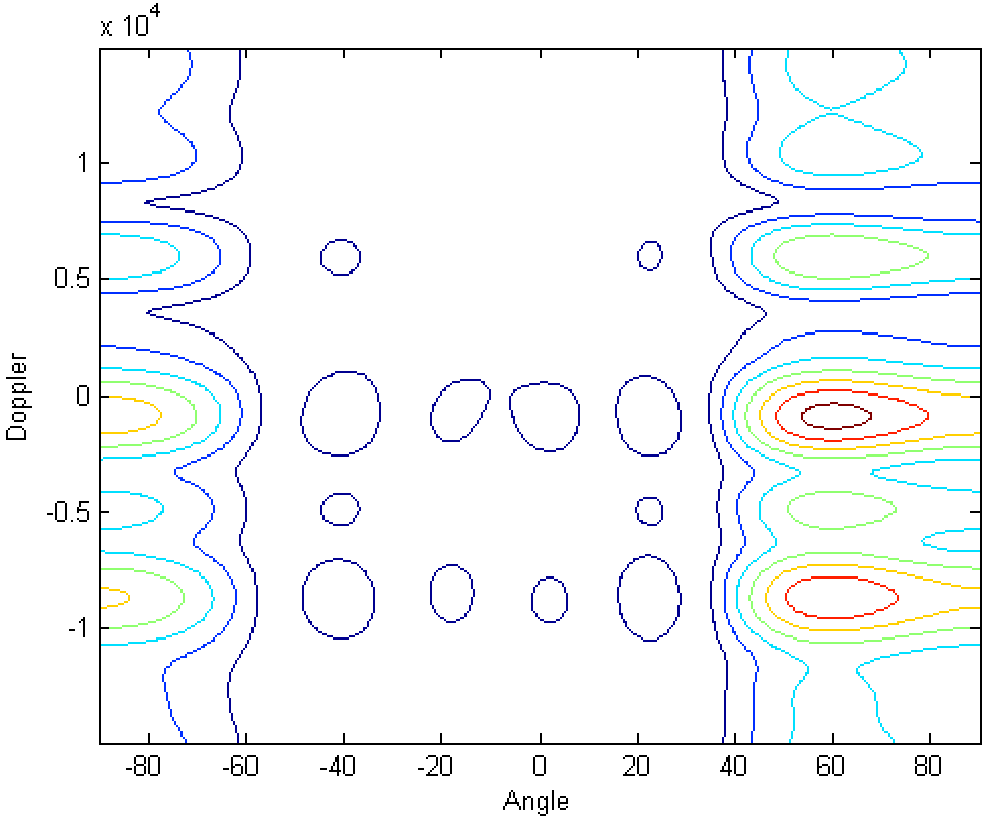
Calculate the angle-Doppler response of the 190th cell of a collected data cube.

Load data and construct AngleDopplerResponse System object

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

Plot Angle-Doppler response

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



Algorithms

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

References

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

See Also

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

phased.AngleDopplerResponse.clone

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.

phased.AngleDopplerResponse.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.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. `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'

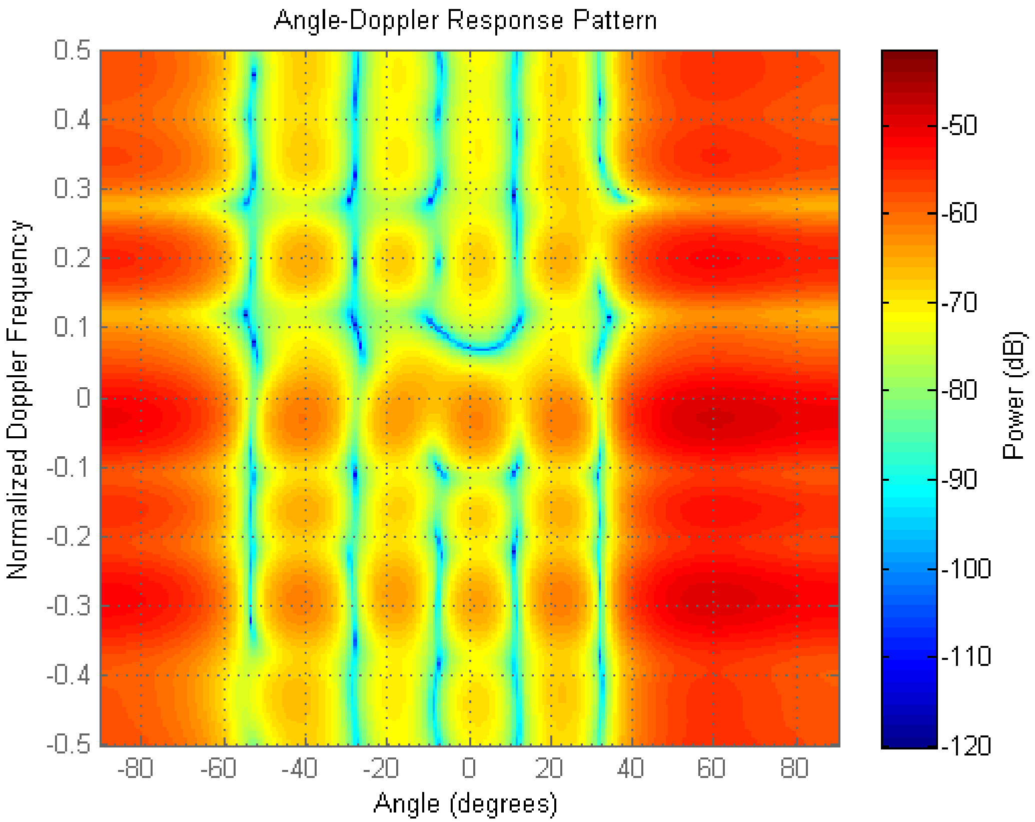
The unit of the plot. Valid values are `'db'`, `'mag'`, and `'pow'`.

Default: `'db'`

Plot Angle-Doppler Response

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

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



See Also `uv2azel` | `phitheta2azel`

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.

---

phased.AngleDopplerResponse.step

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.

Calculate Angle-Doppler response

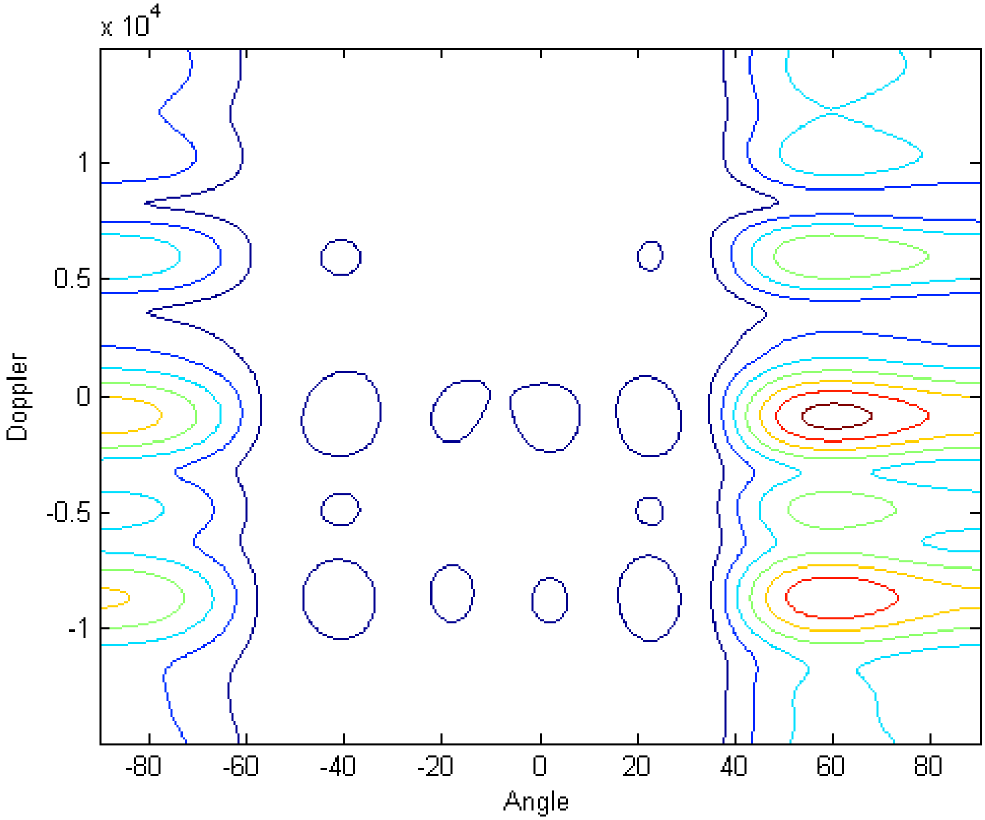
Calculate the angle-Doppler response of the 190th cell of a collected data cube.

Load data and construct AngleDopplerResponse System object

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

Plot Angle-Doppler response

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



Algorithms

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

References

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

See Also

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

phased.ArrayGain

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 1-74.
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

Sensor array

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

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

WeightsInputPort

Add input to specify weights

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

Default: `false`

Methods

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:

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

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

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`.

phased.ArrayGain.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.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.

phased.ArrayGain.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 `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:

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

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

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 1-87.
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`

EnablePolarization

Enable polarization simulation

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

Default: `false`

Methods

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

Plot Array Response

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

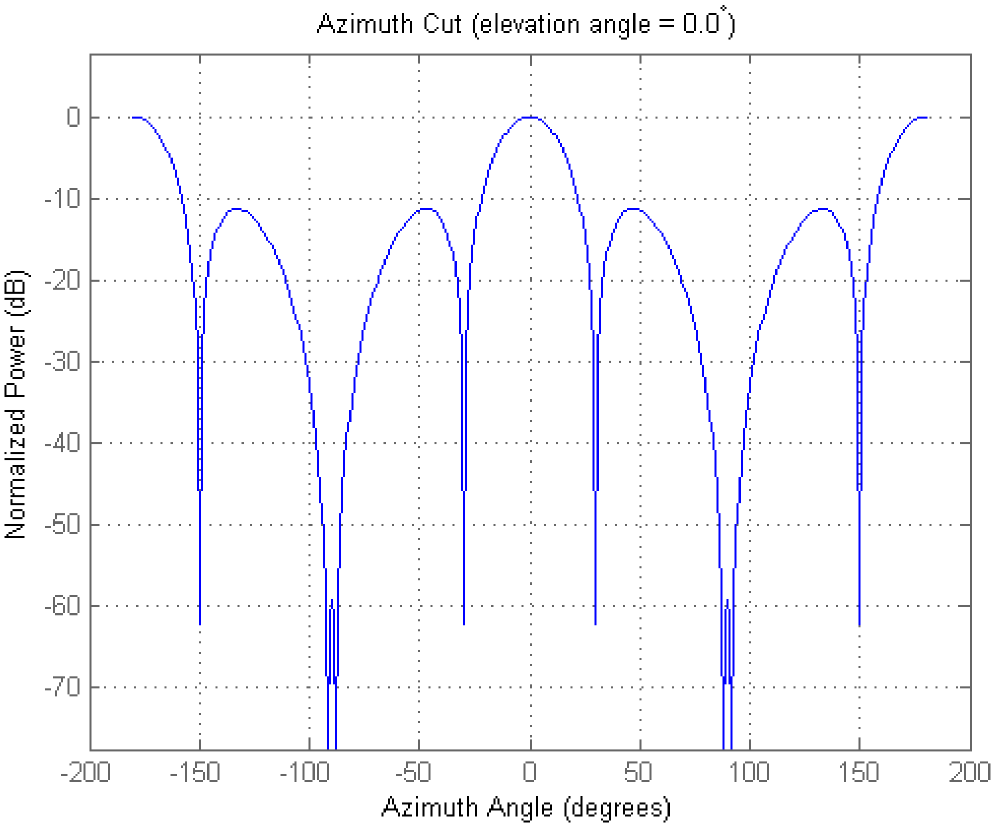
Construct ULA and ArrayResponse System objects

```
ha = phased.ULA(4);
har = phased.ArrayResponse('SensorArray',ha);
resp = step(har,3e8,[30;20]);
```

Plot the array response in dB

By default, the plot has a normalized power and is taken as an azimuth cut at 0 degrees elevation.

```
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.

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.ArrayResponse.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 `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.

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.ArrayResponse.step

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

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

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

Examples

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 1-100.

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

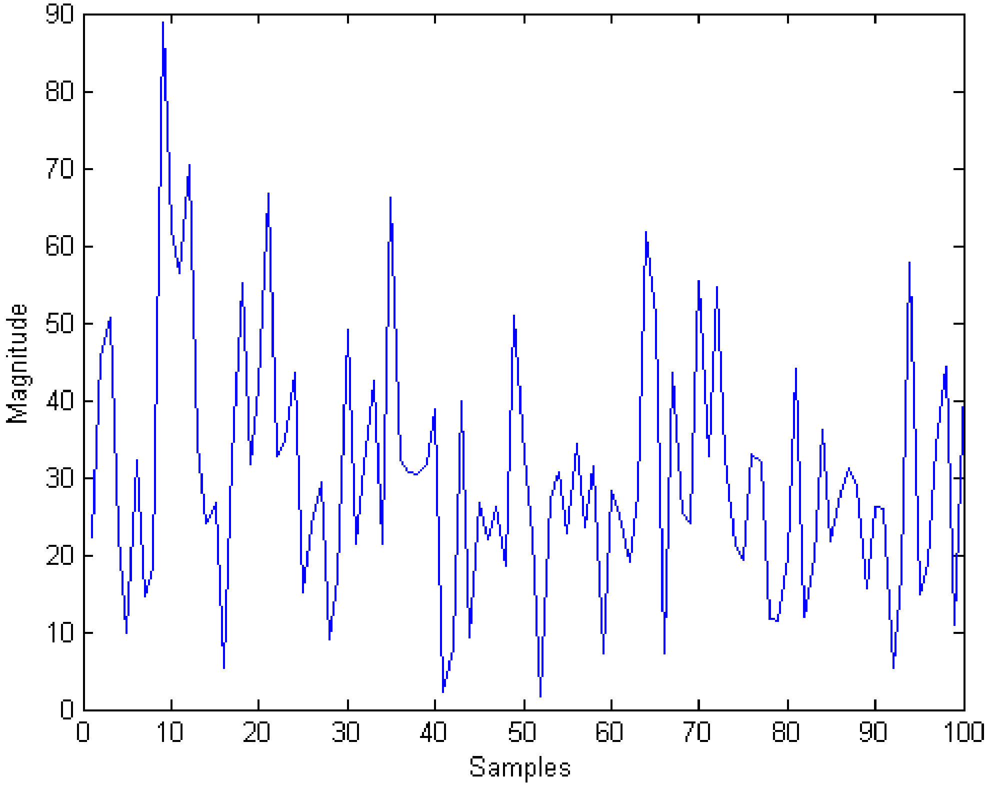
clone	Create barrage jammer object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties

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

Plot Barrage Jammer Output

Create a barrage jammer with an effective radiated power of 1000W. Then plot the magnitude of the jammer 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.

phased.BarrageJammer.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.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.

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'`.

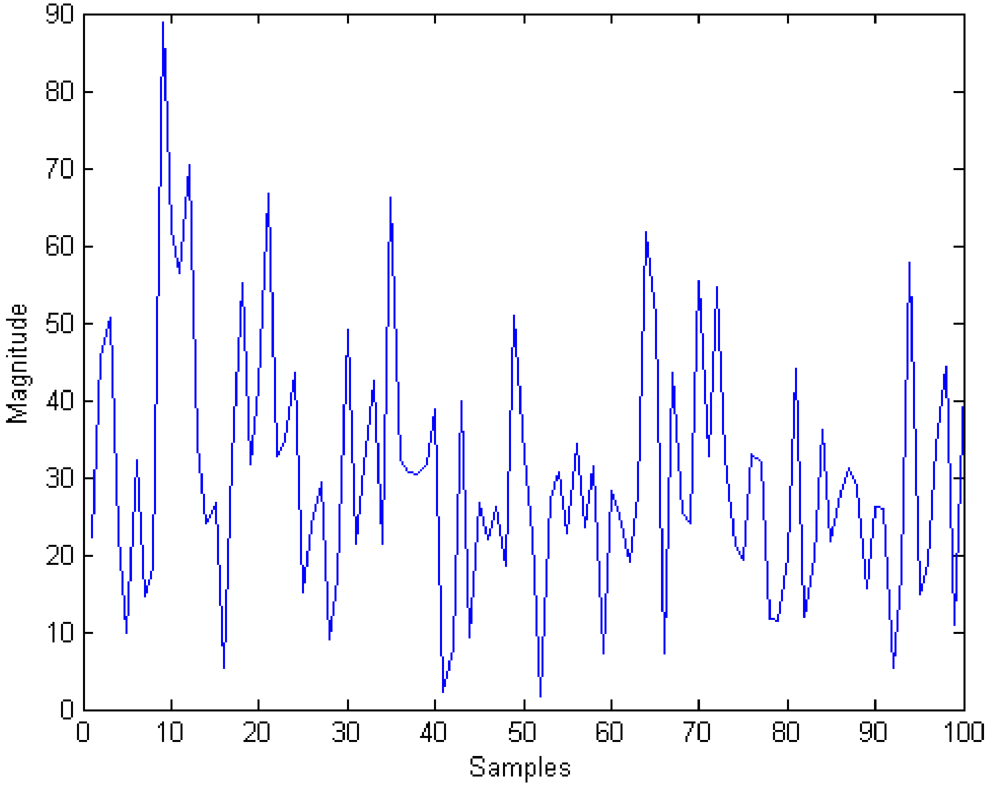
`Y = step(H,N)` returns the jamming signal with length `N`. This syntax is available when the `SamplesPerFrameSource` property is `'Input port'`.

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.

Plot Barrage Jammer Output

Create a barrage jammer with an effective radiated power of 1000W. Then plot the magnitude of the jammer output. Your plot might vary because of random numbers.

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


Magnitude
Samples
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 1-113.
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

Estimate Directions of Arrival of Two Signals

Create the signals and solve for the DOA's

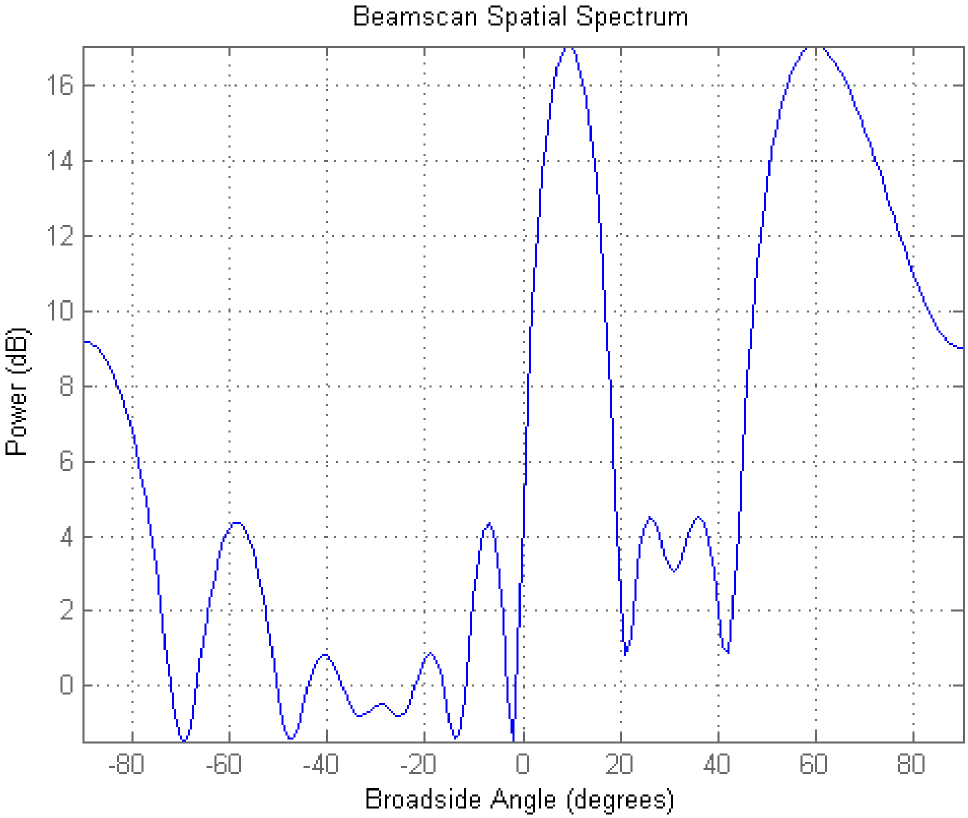
Estimate the DOA's of two signals received by a 10-element ULA with element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 60 degrees in azimuth and -5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

```
doas =

    9.5829   60.3813
```

Plot the beamscan spectrum

```
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.

phased.BeamscanEstimator.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 `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. `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'`

Estimate Directions of Arrival of Two Signals

Create the signals and solve for the DOA's

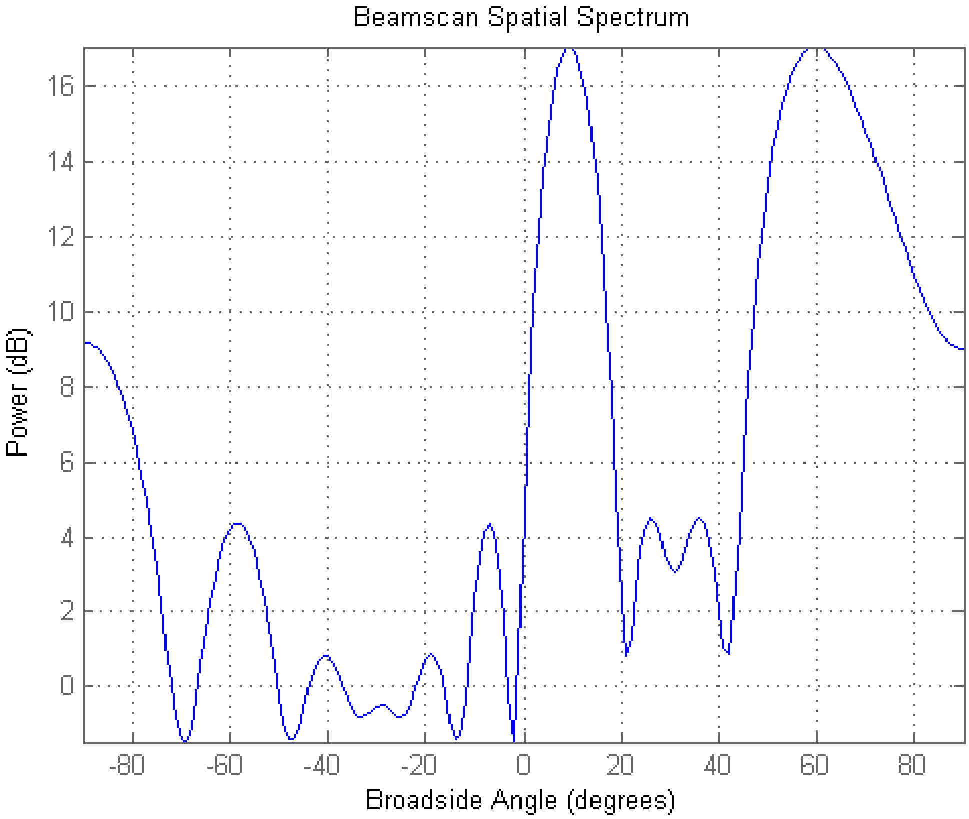
Estimate the DOA's of two signals received by a 10-element ULA with element spacing of 1 meter. The antenna operating frequency is 150 MHz. The actual direction of the first signal is 10 degrees in azimuth and 20 degrees in elevation. The direction of the second signal is 60 degrees in azimuth and -5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

```
doas =

    9.5829   60.3813
```

Plot the beamscan spectrum

```
plotSpectrum(hdoa);
```


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)

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.BeamscanEstimator.reset

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`

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 1-129.
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

Estimate the DOAs of Two Signals

Create the signals and solve for the DOA's

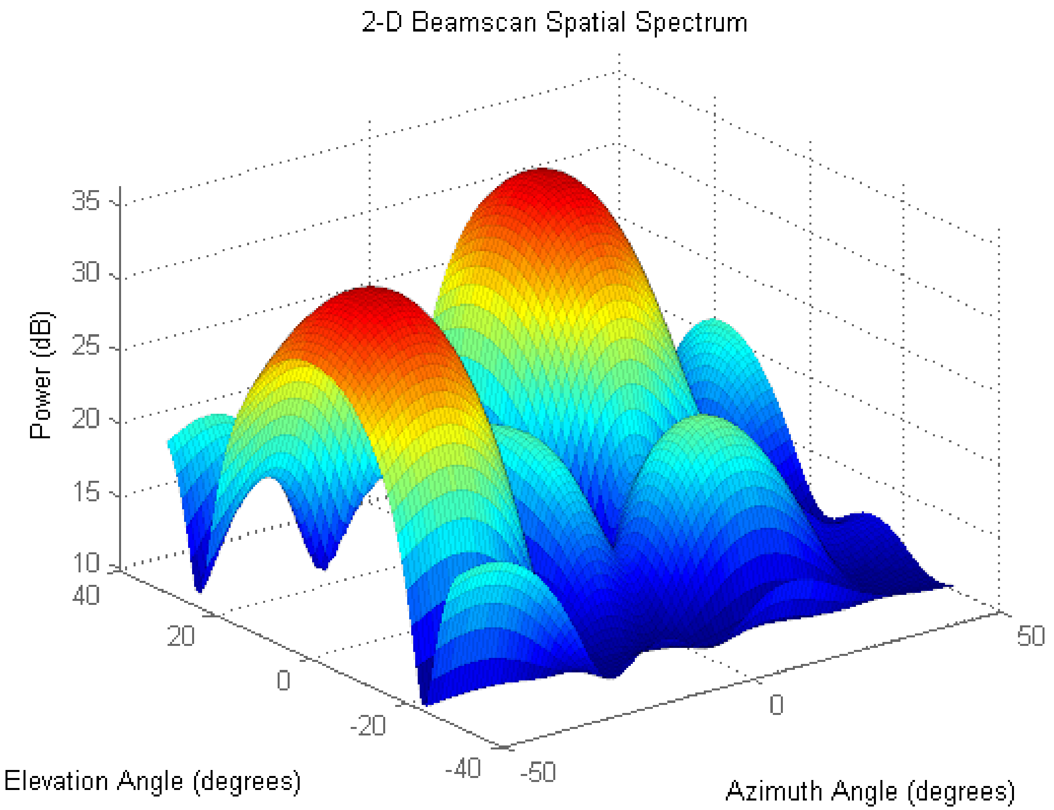
Estimate the DOAs of two signals received by a 50-element URA with a rectangular lattice. The antenna operating frequency is 150 MHz. The actual direction of the first signal is -37 degrees in azimuth and 0 degrees in elevation. The direction of the second signal is 17 degrees in azimuth and 20 degrees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

```
doas =

   -37    17
     0    20
```

Plot the beamscan spatial spectrum

```
plotSpectrum(hdoa);
```



References

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

See Also

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

phased.BeamscanEstimator2D.clone

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.

phased.BeamscanEstimator2D.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. `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'`

Estimate the DOAs of Two Signals

Create the signals and solve for the DOA's

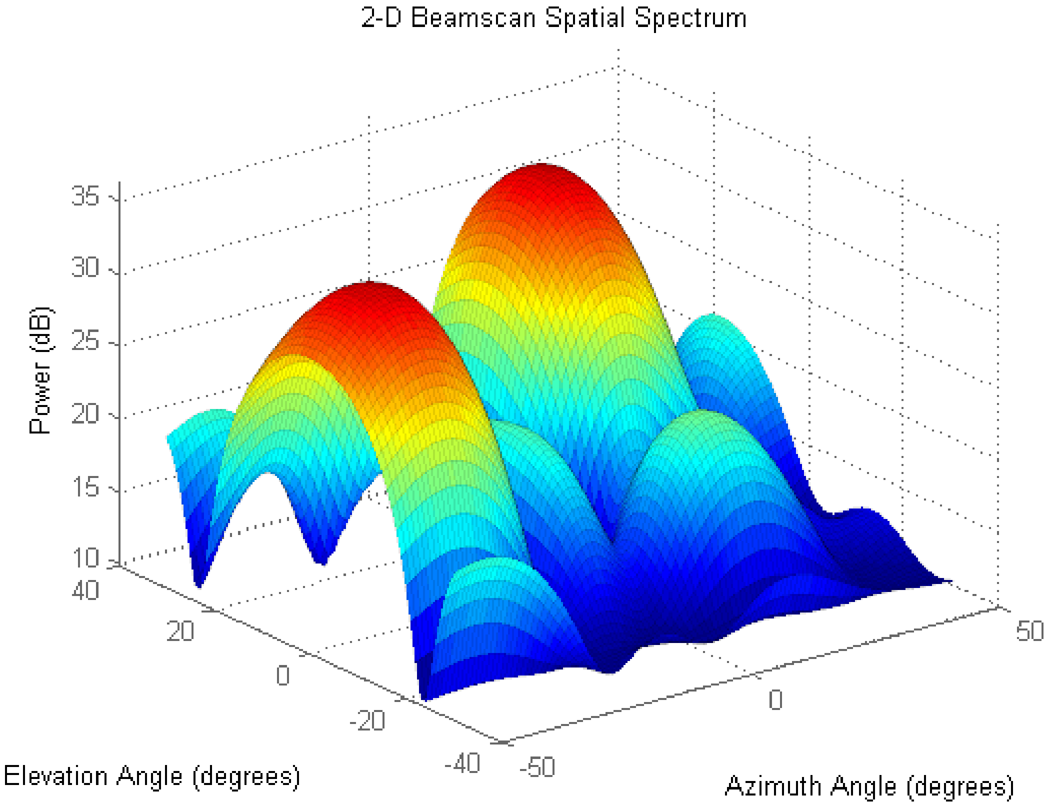
Estimate the DOAs of two signals received by a 50-element URA with a rectangular lattice. The antenna operating frequency is 150 MHz. The actual direction of the first signal is -37 degrees in azimuth and 0 degrees in elevation. The direction of the second signal is 17 degrees in azimuth and 20 degrees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

```
doas =

   -37    17
     0    20
```

Plot the beamscan spatial spectrum

```
plotSpectrum(hdoa);
```


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

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`

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 1-145.

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

`broadside2az` | `phased.ESPRITEstimator`

phased.BeamspaceESPRITEstimator.clone

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.

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 1-157.
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 *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 further details, see [1].

References

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

See Also

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

phased.CFARDetector.clone

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`.

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.CFARDetector.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 `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.

phased.CFARDetector.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 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 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 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 1-170.
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

Sensor element or sensor array

Sensor element or sensor array specified as a System object in the Phased Array System Toolbox™. A sensor array can contain subarrays.

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

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`

EnablePolarization

EnablePolarization

Set this property to `true` to simulate the collection of polarized waves. Set this property to `false` to ignore polarization. This property applies when the sensor specified in the `Sensor` property is capable of simulating polarization.

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`

phased.Collector.clone

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`.

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.Collector.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 `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.

phased.Collector.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

Collect signals

Syntax

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

Description

`Y = step(H,X,ANG)` collects signals `X` arriving from directions `ANG`. The collection process depends on the `Wavefront` property of `H`, as follows:

- If `Wavefront` has the value `'Plane'`, each collecting element collects all the far field signals in `X`. Each column of `Y` contains the output of the corresponding element in response to all the signals in `X`.
- If `Wavefront` has the value `'Unspecified'`, each collecting element collects only one impinging signal from `X`. Each column of `Y` contains the output of the corresponding element in response to the corresponding column of `X`. The `'Unspecified'` option is available when the `Sensor` property of `H` does not contain subarrays.

`Y = step(H,X,ANG,LAXES)` uses `LAXES` as the local coordinate system axes directions. This syntax is available when you set the `EnablePolarization` property to `true`.

`Y = step(H,X,ANG,WEIGHTS)` uses `WEIGHTS` as the weight vector. This syntax is available when you set the `WeightsInputPort` property to `true`.

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

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

Note The object performs an initialization the first time the `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.

- If the `EnablePolarization` property value is set to `false`, `X` is a matrix. The number of columns of the matrix equals the number of separate signals.
- If the `EnablePolarization` property value is set to `true`, `X` is a row vector of MATLAB `struct` type. The dimension of the `struct` array equals the number of separate signals. Each

`struct` member contains three column-vector fields, `X`, `Y`, and `Z`, representing the *x*, *y*, and *z* components of the polarized wave vector signals in the global coordinate system.

ANG

Incident directions of signals, specified as a two-row matrix. Each column specifies the incident direction of the corresponding column of `X`. Each column of `ANG` has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

LAXES

Local coordinate system. `LAXES` is a 3-by-3 matrix whose columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively. Each axis is specified in terms of `[x;y;z]` with respect to the global coordinate system. This argument is only used when the `EnablePolarization` property is set to `true`.

WEIGHTS

Vector of weights. `WEIGHTS` is a column vector of length M, where M is the number of collecting elements.

Default: `ones(M,1)`

STEERANGLE

Subarray steering angle, specified as a length-2 column vector. The vector has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

Output Arguments

Y

Collected signals. Each column of `Y` contains the output of the corresponding element. The output is the response to all the signals in `X`, or one signal in `X`, depending on the `Wavefront` property of `H`.

Examples

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`

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 1-183.
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. Angle units are degrees. The value assigned to `ElementNormal` must be either a 2-by-N matrix or a 2-by-1 column vector. The variable N indicates the number of elements in the array. If the value of `ElementNormal` is a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive x-axis with the direction normal to the conformal array. If the value of `ElementNormal` is a 2-by-1 column vector, it specifies the same pointing direction for all elements in the array.

You can use the `ElementPosition` and `ElementNormal` properties to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation.

However, you cannot use transformations that require rotation about the normal.

Default: `[0; 0]`

Taper

Element taper or weighting

Element taper or weighting specified as a complex scalar or 1-by-N complex-valued vector. Weights are applied to each element in the sensor array. N is the number of elements along in the array as determined by the size of the `ElementPosition` property. If the `Taper` parameter is a scalar, identical weights will be applied to each element. If the value of `Taper` is a vector, each weight will be applied to the corresponding element.

Default: 1

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
getTaper	Array element tapers
isLocked	Locked status for input attributes and nontunable properties
isPolarizationCapable	Polarization capability

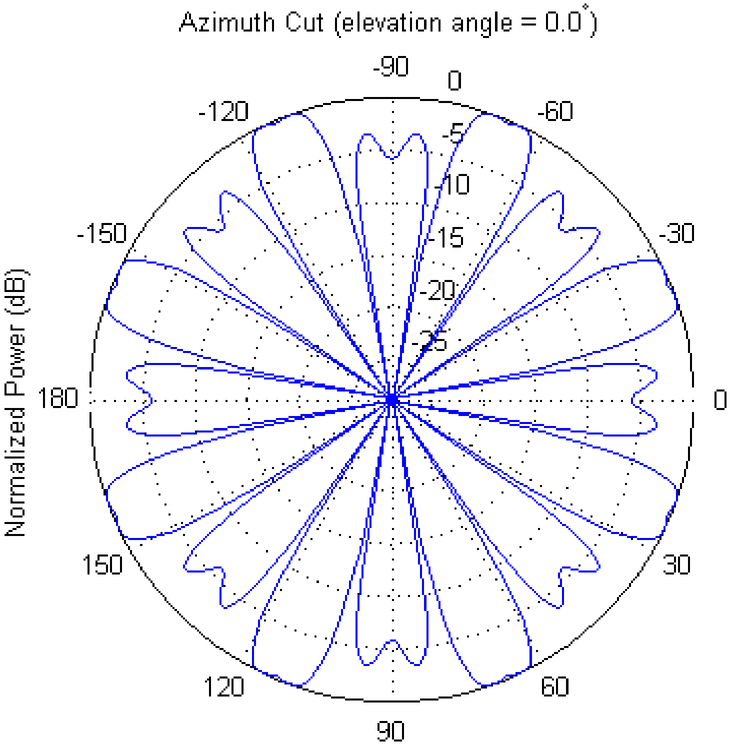
plotResponse	Plot response pattern of array
release	Allow property value and input characteristics changes
step	Output responses of array elements
viewArray	View array geometry

Examples

Plot Response of 8-Element Uniform Circular Array

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');
```



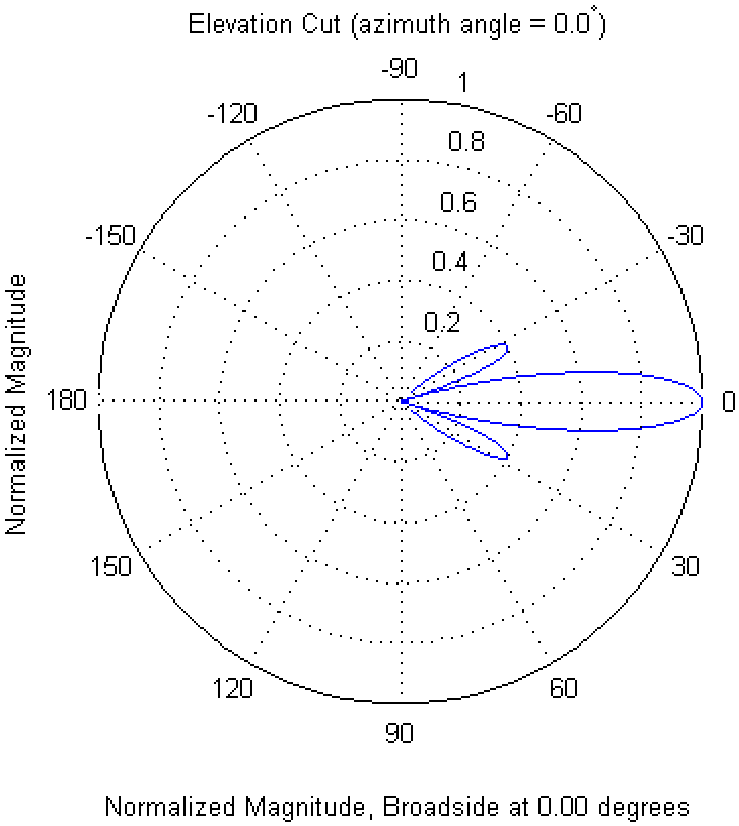
Normalized Power (dB), Broadside at 0.00 degrees

Plot Response of 31-Element Uniform Circular Sonar Array

Construct a 31-element uniform circular sonar array (UCA), one meter in diameter. Using the `ElevationAngles` parameter, restrict the display to +/-40 degrees in 0.1 degree increments. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s.

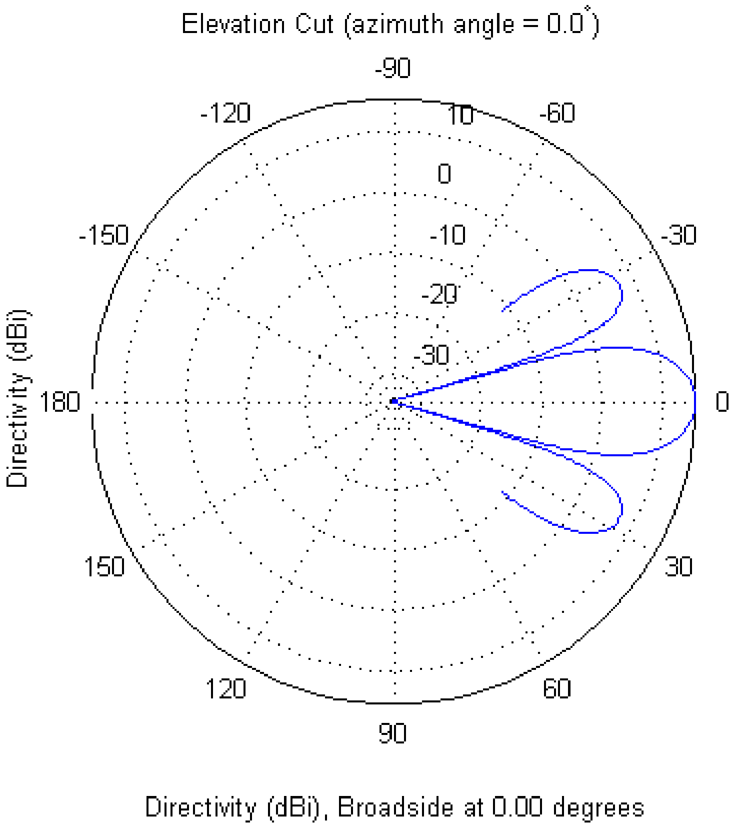
```
N = 31;
theta = (0:N-1)*360/N-180;
```

```
Radius = 0.5;
s_mic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[0,10000],'BackBaffled',true);
s_array = phased.ConformalArray('Element',s_mic,...
    'ElementPosition',Radius*[zeros(1,N);cosd(theta);sind(theta)],...
    'ElementNormal',[ones(1,N);zeros(1,N)]);
fc = 4000;
c = 1500.0;
plotResponse(s_array,fc,c,'RespCut','El',...
    'Format','Polar','Unit','mag',...
    'ElevationAngles',[-40:0.1:40]);
```



Plot the directivity.

```
plotResponse(s_array,fc,c,'RespCut','El',...
    'Format','Polar','Unit','dbi',...
    'ElevationAngles',[-40:0.1:40]);
```



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

phased.ConformalArray.clone

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`.

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`

phased.ConformalArray.getElementPosition

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)
```

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)
```

phased.ConformalArray.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.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.

phased.ConformalArray.getTaper

Purpose

Array element tapers

Syntax

`wts = getTaper(h)`

Description

`wts = getTaper(h)` returns the tapers applied to each element of a conformal array, `h`. Tapers are often referred to as weights.

Input Arguments

h - Conformal array

`phased.ConformalArray` System object

Conformal array specified as a `phased.ConformalArray` System object.

Output Arguments

wts - Array element tapers

N-by-1 complex-valued vector

Array element tapers returned as an N-by-1, complex-valued vector, where N is the number of elements in the array.

Examples

Create and View a Tapered Array

Create a two-ring tapered disk array

Create a two-ring disk array and set the taper values on the outer ring to be smaller than those on the inner ring.

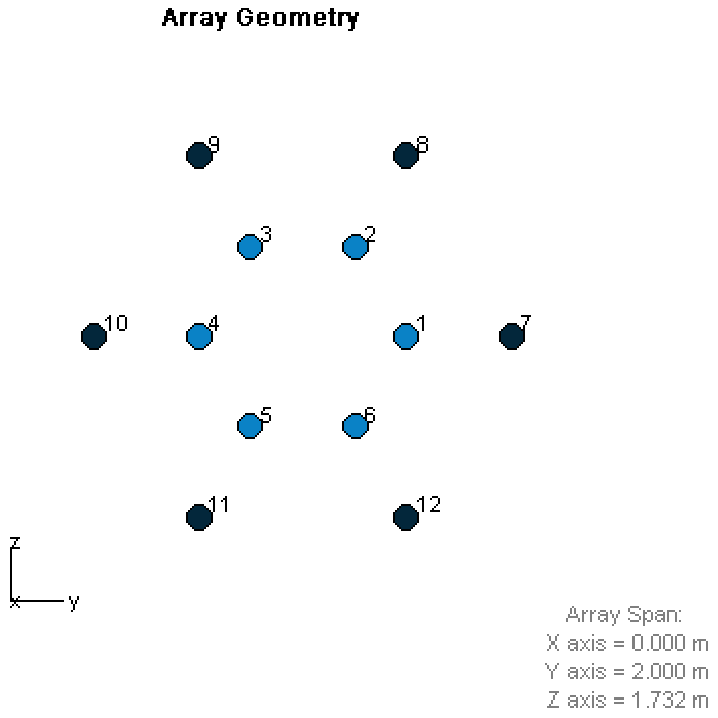
```
elemAngles = ([0:5]*360/6);
elemPosInner = 0.5*[zeros(size(elemAngles));...
    cosd(elemAngles);...
    sind(elemAngles)];
elemPosOuter = [zeros(size(elemAngles));...
    cosd(elemAngles);...
    sind(elemAngles)];
elemNorms = repmat([0;0],1,12);
taper =  [ones(size(elemAngles)),0.3*ones(size(elemAngles))];
ha = phased.ConformalArray(...
    [elemPosInner,elemPosOuter],elemNorms,'Taper',taper);
```

Display the taper values

```
w = getTaper(ha)


w =

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    0.3000
    0.3000
    0.3000
    0.3000
    0.3000
    0.3000
```

View the array

```
viewArray(ha,'ShowTaper',true,'ShowIndex','all');
```


Array Geometry
z
x
y
Array Span:
X axis = 0.000 m
Y axis = 2.000 m
Z axis = 1.732 m

phased.ConformalArray.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 `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.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h - Conformal array

Conformal array specified as a `phased.ConformalArray` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

Conformal Array of Short-dipole Antenna Elements Supports Polarization

Show that a circular conformal array of `phased.ShortDipoleAntennaElement` antenna elements supports polarization.

```
N = 8; azang = (0:N-1)*360/N-180;
h = phased.ShortDipoleAntennaElement;
ha = phased.ConformalArray(...
    'Element',h,'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],..
    'ElementNormal',[azang;zeros(1,N)]);

isPolarizationCapable(ha)
```

```
ans =

     1
```

The returned value `true (1)` shows that this array supports polarization.

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 specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the array response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'Weights'

Weight values applied to the array, specified as a length-N column vector or N-by-M matrix. The dimension N is the number of elements in the array. The interpretation of M depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-M row vector	Apply one set of weights for the same single frequency or all M frequencies.
N-by-M matrix	Scalar	Apply all of the M different columns in `Weights` for the same single frequency.
	1-by-M row vector	Apply each of the M different columns in `Weights` for the corresponding frequency in `FREQ`.

'AzimuthAngles'

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

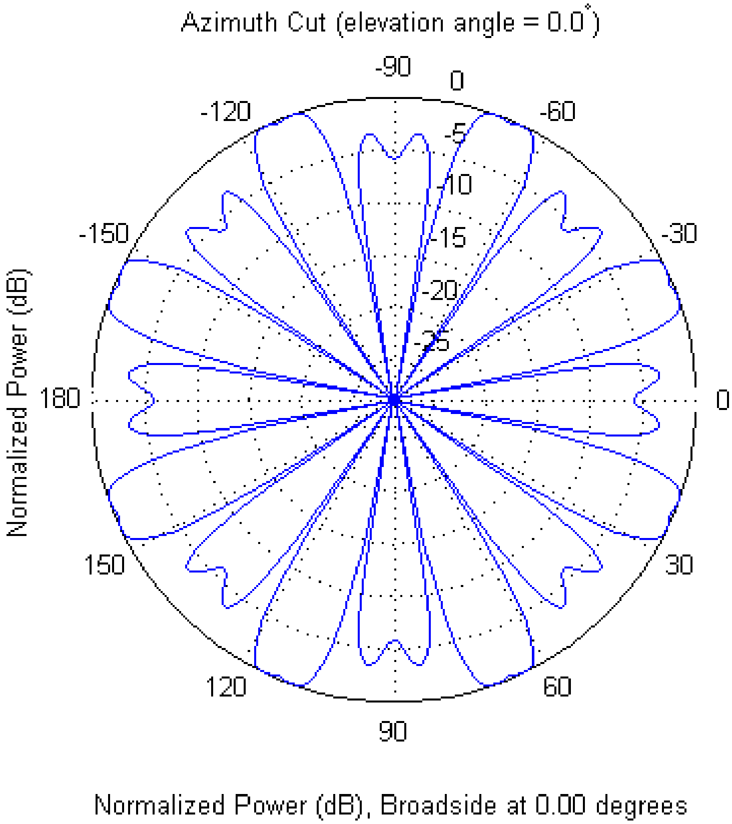
Default: `[-1:0.01:1]`

Examples

Plot Response of 8-Element Uniform Circular Array

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');
```

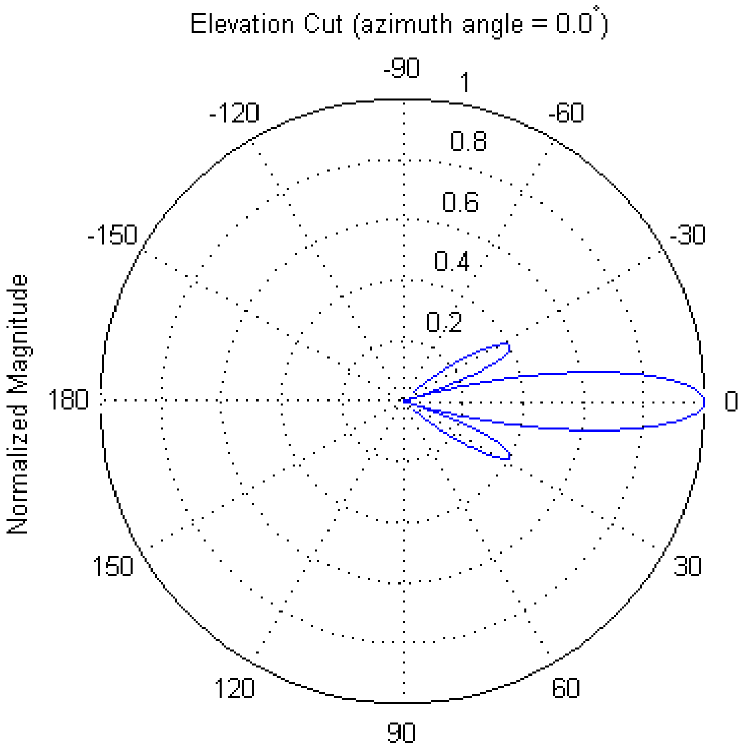


Plot Response of 31-Element Uniform Circular Sonar Array

Construct a 31-element uniform circular sonar array (UCA), one meter in diameter. Using the `ElevationAngles` parameter, restrict the display to +/-40 degrees in 0.1 degree increments. Assume the operating frequency is 4 kHz. A typical value for the speed of sound in seawater is 1500.0 m/s.

```
N = 31;
theta = (0:N-1)*360/N-180;
```

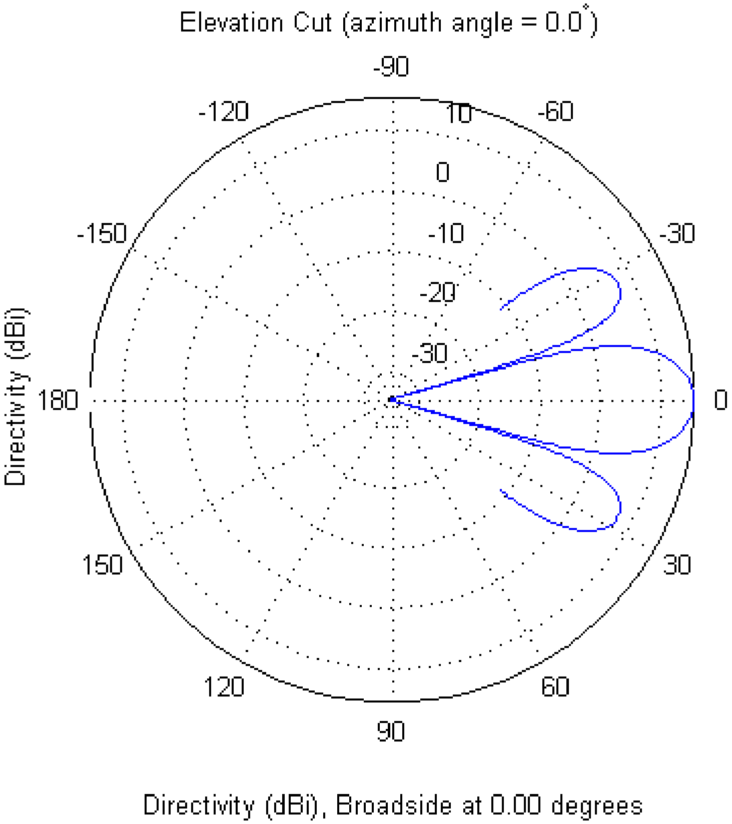
```
Radius = 0.5;
s_mic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[0,10000],'BackBaffled',true);
s_array = phased.ConformalArray('Element',s_mic,...
    'ElementPosition',Radius*[zeros(1,N);cosd(theta);sind(theta)],...
    'ElementNormal',[ones(1,N);zeros(1,N)]);
fc = 4000;
c = 1500.0;
plotResponse(s_array,fc,c,'RespCut','El',...
    'Format','Polar','Unit','mag',...
    'ElevationAngles',[-40:0.1:40]);
```



Normalized Magnitude, Broadside at 0.00 degrees

Plot the directivity.

```
plotResponse(s_array,fc,c,'RespCut','El',...
    'Format','Polar','Unit','dbi',...
    'ElevationAngles',[-40:0.1:40]);
```



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`

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions N-by-M-by-L. N is the number of elements in the array. The dimension M is the number of angles specified in `ANG`. L is the number of frequencies specified in `FREQ`. For any element, the columns of `RESP` contain the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.
- If the array is capable of supporting polarization, the voltage response, `RESP`, is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions N-by-M-by-L. N is the number of elements in the array, and M is the number of angles specified in `ANG`. L is the number of frequencies specified in `FREQ`. Each column of `RESP` contains the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.

Examples

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);

resp =

     1
     1
     1
     1
     1
     1
     1
     1
```

See Also

`uv2azel` | `phitheta2azel`

phased.ConformalArray.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. `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`

'ShowTaper'

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

'Title'

String specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

Handle of array elements in figure window.

Examples

View Uniform Circular Array

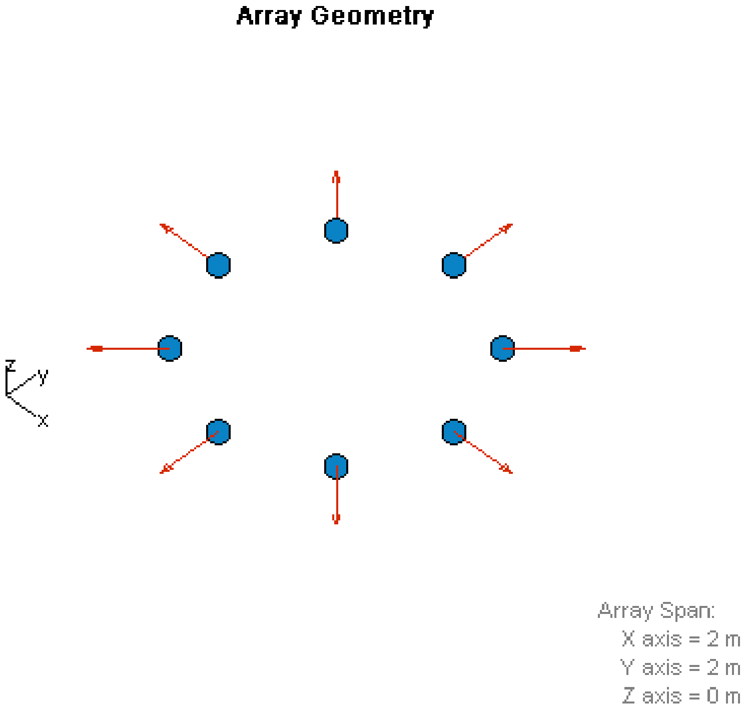
Display the element positions and normal directions of all elements of an 8-element uniform circular array.

Create the uniform circular array

```
N = 8;
azang = (0:N-1)*360/N - 180;
ha = phased.ConformalArray(...
    'ElementPosition',[cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)]);
```

Display the positions and normal directions of the elements

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



See Also `phased.ArrayResponse`

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 1-223.
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 radar system

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 clutter simulation object

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 clutter return

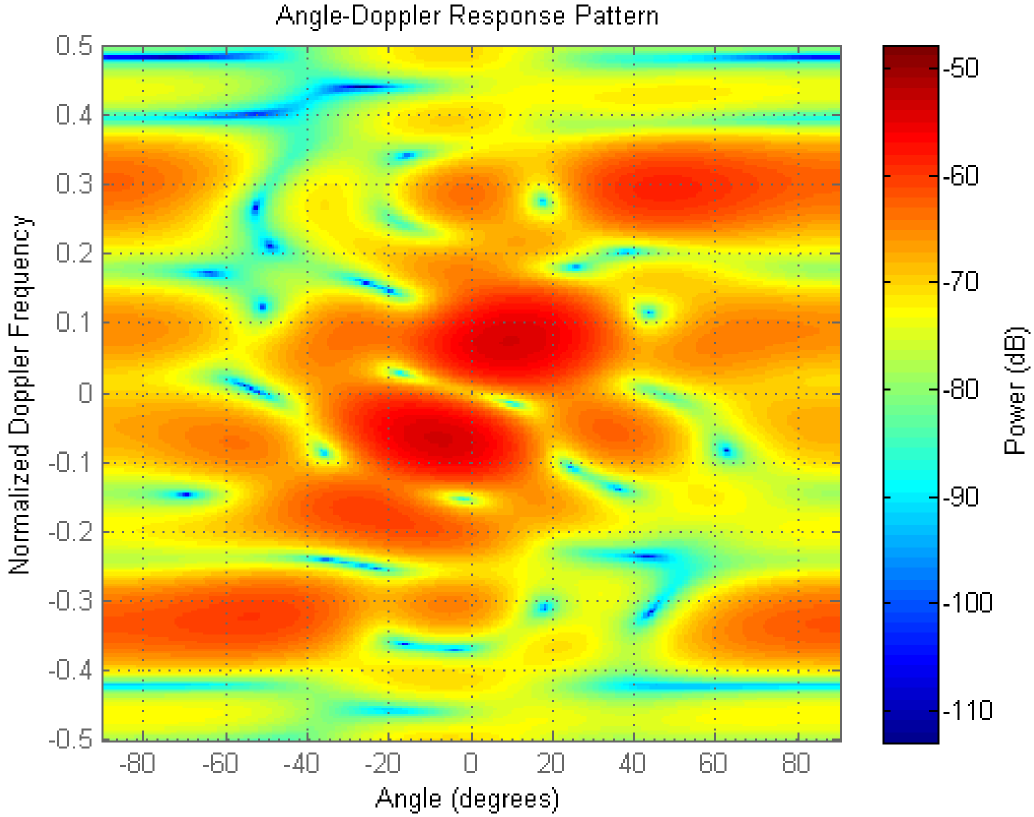
Simulate the clutter return for 10 pulses.

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

Plot angle-Doppler response

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 radar system

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 clutter simulation object

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 clutter return

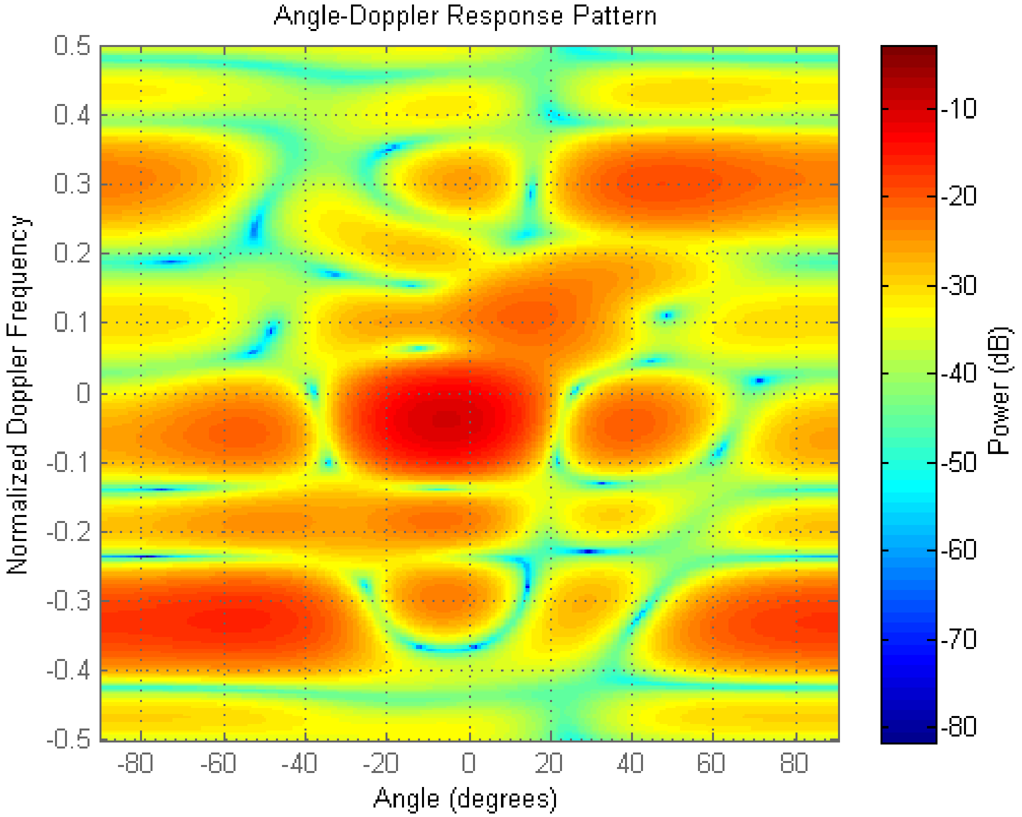
Simulate the clutter return for 10 pulses. At each step, pass the transmit signal as an input argument. The software automatically computes the effective transmitted power of the signal. The transmit signal is a rectangular waveform with a pulse width of 2 microseconds.

```
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 angle-Doppler response

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"

phased.ConstantGammaClutter.clone

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)`

`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.

phased.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.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);
```

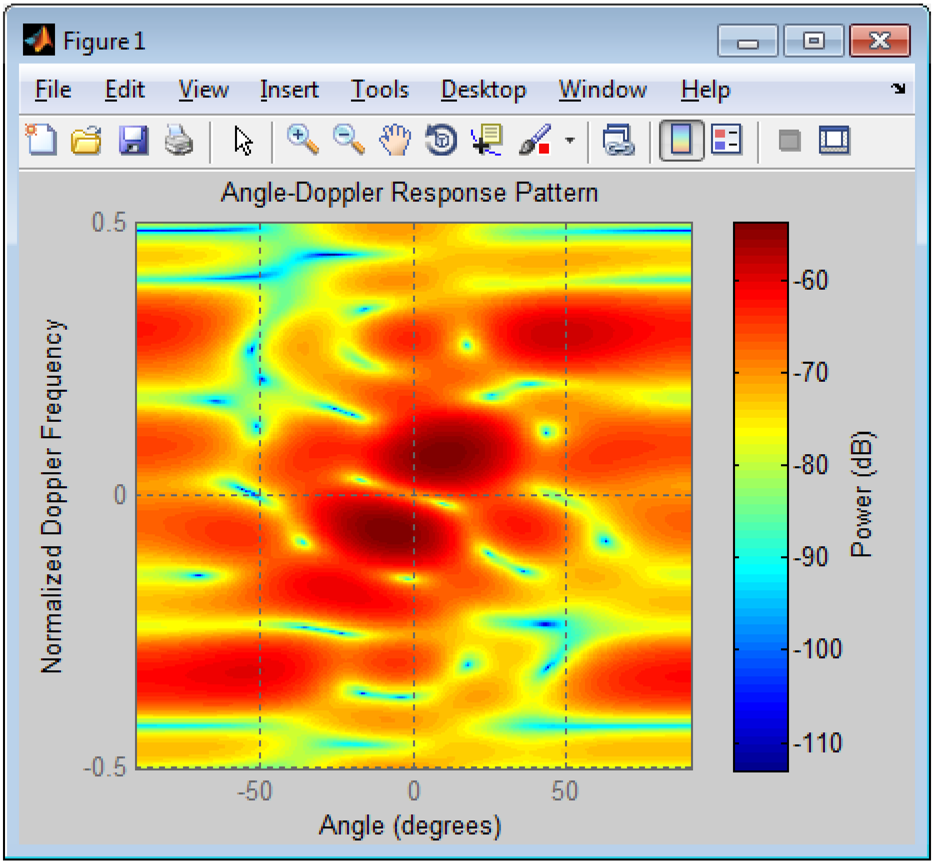


Figure 1
File Edit View Insert Tools Desktop Window Help
Angle-Doppler Response Pattern
Normalized Doppler Frequency
0.5
0
-0.5
-50
0
50
Angle (degrees)
-60
-70
-80
-90
-100
-110
Power (dB)

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);
```

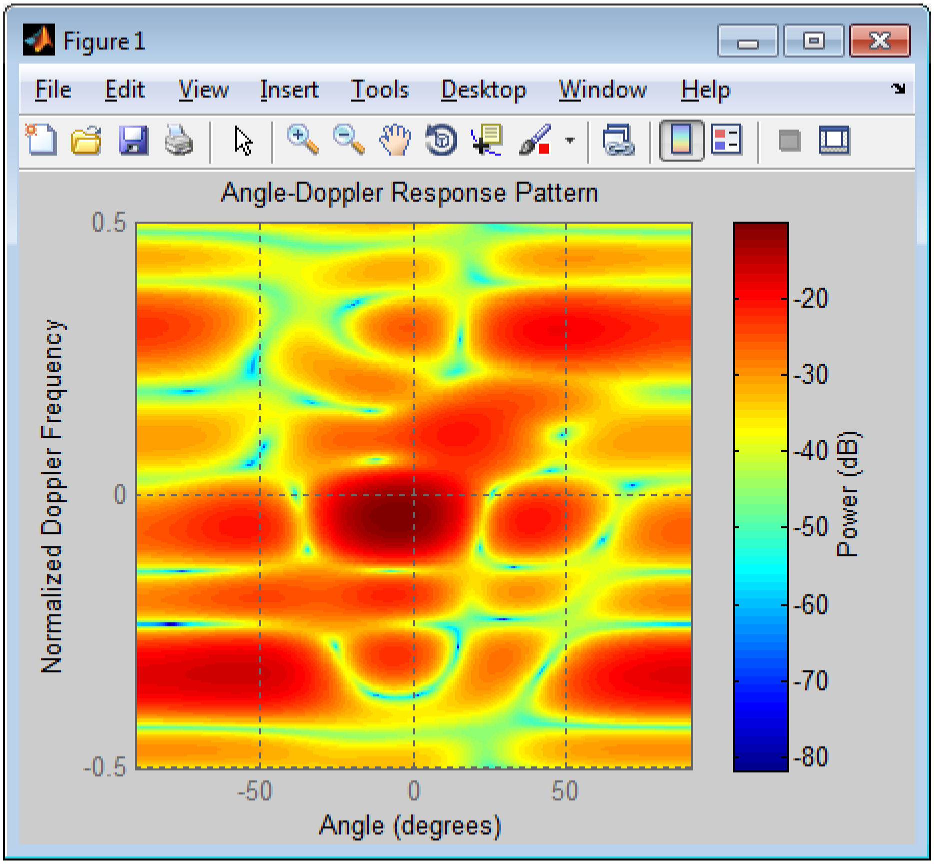
Simulate the clutter return for 10 pulses. At each step, pass the transmit signal as an input argument. The software automatically computes the effective transmitted power of the signal. The transmit signal is a rectangular waveform with a pulse width of 2 μs.

```
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"

phased.CosineAntennaElement

Purpose

Cosine antenna element

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 1-252.
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.

This antenna element is not capable of supporting polarization.

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 Hz) of the antenna element as a 1-by-2 row vector in the form `[LowerBound HigherBound]`. The antenna element has no response outside the specified frequency range.

Default: `[0 1e20]`

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
isPolarizationCapable	Polarization capability
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.

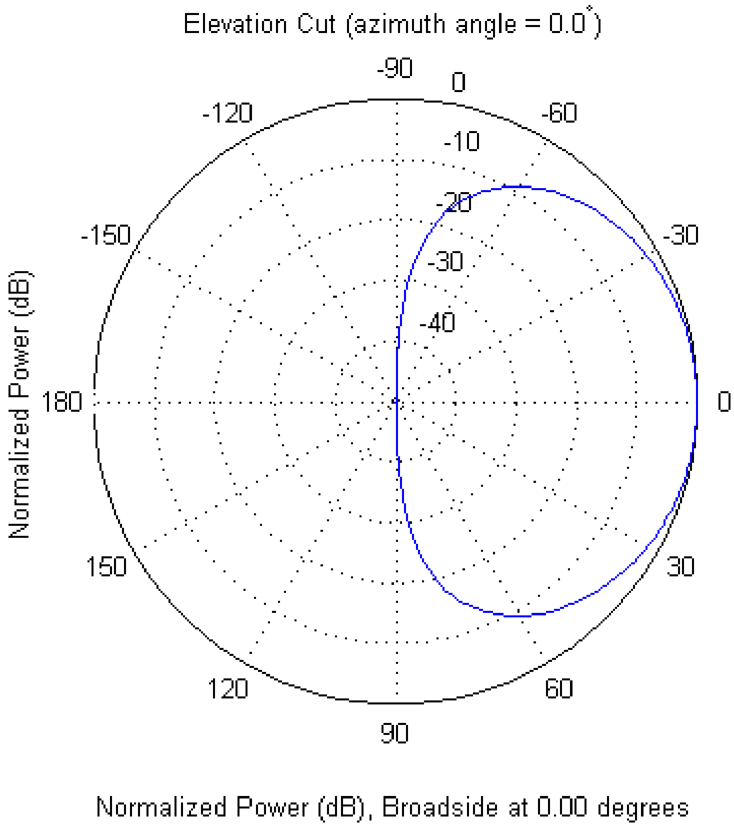
The response is defined for azimuth and elevation angles between –90 and 90 degrees, inclusive. There is no response at the back of a cosine antenna. The cosine response pattern achieves a maximum value of 1 at 0 degrees azimuth and elevation. Raising the response pattern to powers greater than one concentrates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

Examples

Calculate Response of Cosine Antenna

This example shows how to construct a cosine pattern antenna and calculate its response at boresight (0 degrees azimuth and 0 degrees elevation). Assume the antenna works between 800 MHz and 1.2 GHz and its 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.CrossedDipoleAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ShortDipoleAntennaElement` | `phased.ULA` | `phased.URA` | `phased.ConformalArray`

phased.CosineAntennaElement.clone

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.

phased.CosineAntennaElement.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.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.

phased.CosineAntennaElement.isPolarizationCapable

Purpose Polarization capability

Syntax `flag = isPolarizationCapable(h)`

Description `flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.CosineAntennaElement` System object supports polarization. An antenna element supports polarization if it can create or respond to polarized fields. This object does not support polarization.

Input Arguments

h - Cosine antenna element

Cosine antenna element specified as a `phased.CosineAntennaElement` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability flag returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not. Because the `phased.CosineAntennaElement` object does not support polarization, `flag` is always returned as `false`.

Examples

Cosine Antenna Does Not Support Polarization

Create a cosine antenna element using the `phased.CosineAntennaElement` antenna element and show that it does not support polarization.

```
h = phased.CosineAntennaElement('FrequencyRange',[1.0,10]*1e9);
isPolarizationCapable(h)
```

```
ans =

     0
```

The returned value `false (0)` shows that the antenna element does not support polarization.

phased.CosineAntennaElement.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 System object

FREQ

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the antenna response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'AzimuthAngles'

Azimuth angles for plotting element response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting element response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting element response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting element response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

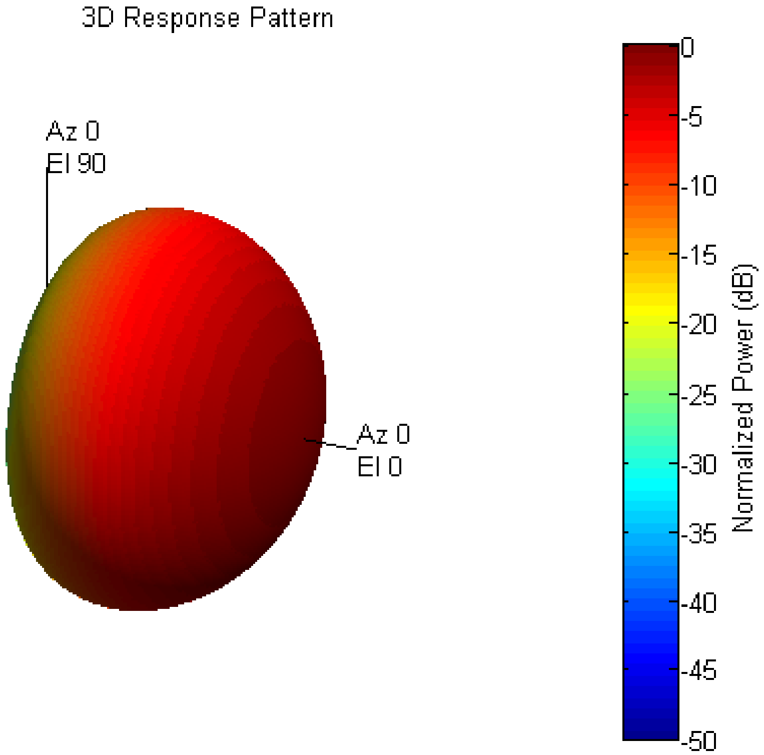
Default: `[-1:0.01:1]`

Examples

Plot 3-D Polar Response of Cosine Antenna

This example shows how to plot the 3-D polar response of a cosine antenna element. Construct a cosine antenna element using default parameters. Assume the antenna operating frequency is 1 GHz. Then, plot the antenna's response in 3-D polar format.

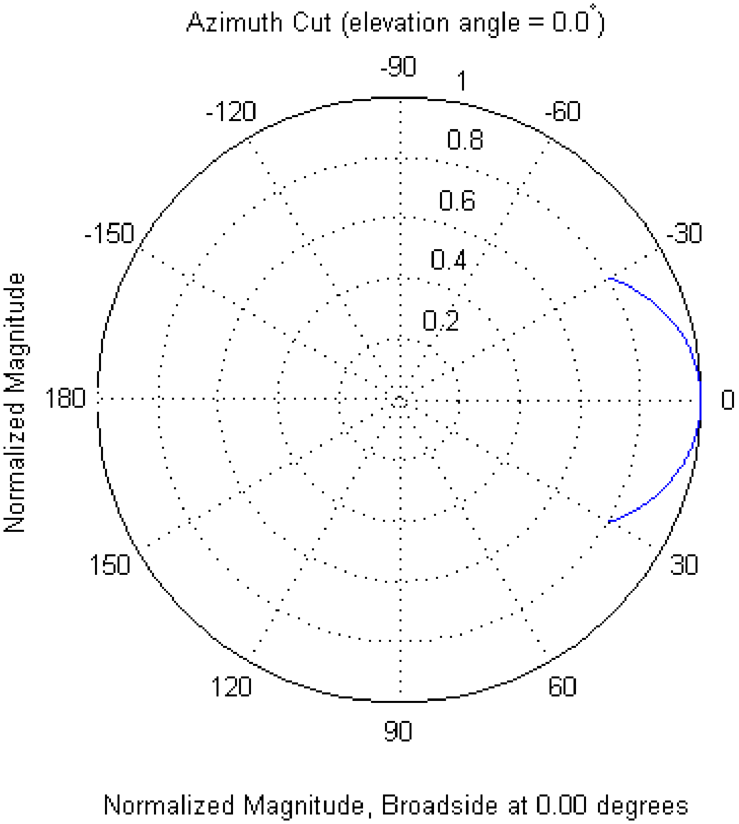
```
hcos = phased.CosineAntennaElement;
plotResponse(hcos,1e9,'Format','Polar','RespCut','3D');
```



Plot Azimuth-Cut of Cosine Antenna Response

This example shows how to plot an azimuth-cut of the cosine antenna response. Construct a cosine antenna element using default parameters. Assume the antenna operating frequency is 1 GHz. Restrict the response to the range of azimuth angles from -30 to 30 degrees in 0.1 degree increments. The default elevation angle is 0 degrees.

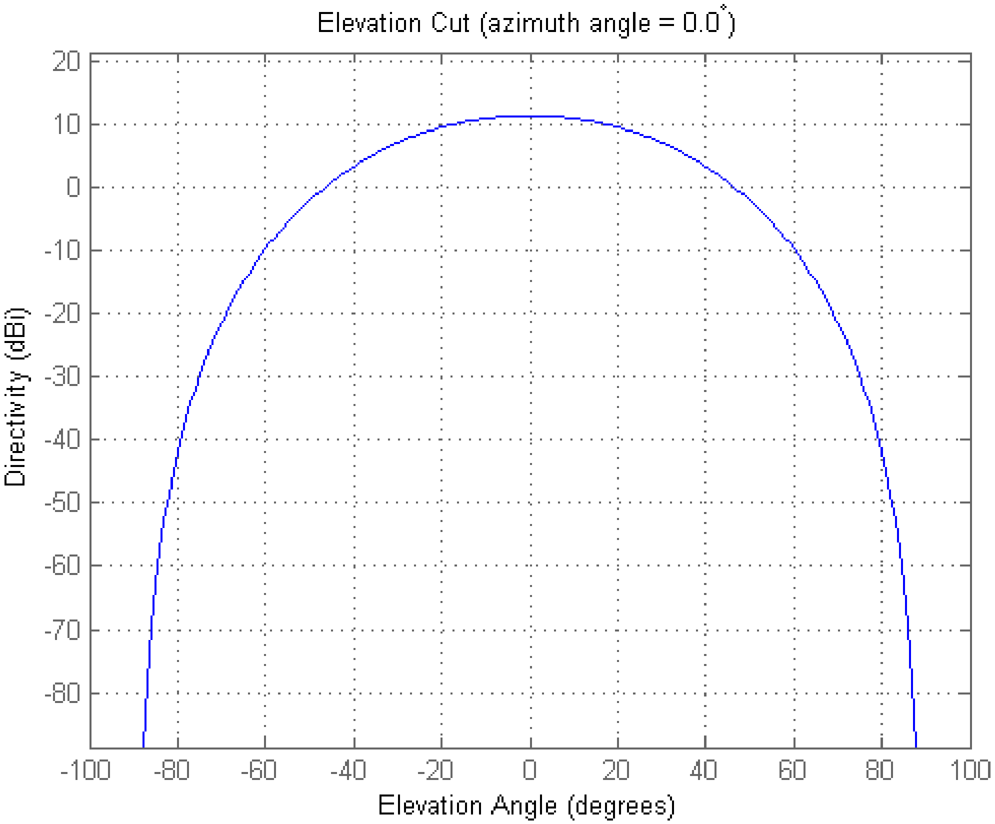
```
hcos = phased.CosineAntennaElement;
plotResponse(hcos,1e9,'Format','Polar','RespCut','Az',...
    'AzimuthAngles',[-30:0.1:30],'Unit','mag');
```



Plot Directivity of Cosine Antenna

This example shows how to construct a cosine-pattern antenna and plot an elevation cut of its directivity. Assume the antenna works between 1 and 2 GHz and its operating frequency is 1.5 GHz. Set the azimuth angle cosine power to 2.5 and the elevation angle cosine power to 3.5.

```
sCos = phased.CosineAntennaElement('FrequencyRange',...
    [1e9 2e9],'CosinePower',[2.5,3.5]);
plotResponse(sCos,1.5e9,'RespCut','El','Unit','dbi');
```



The directivity is maximum at 0 degrees elevation and attains a value of approximately 12 dB.

See Also

`uv2azel` | `azel2uv`

phased.CosineAntennaElement.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.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 specified as an M-by-L, complex-valued matrix. In this matrix, M represents the number of angles specified in `ANG` while L represents the number of frequencies specified in `FREQ`.

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

Crossed-dipole antenna element

Description

The `phased.CrossedDipoleAntennaElement` System object models a *crossed-dipole antenna element*. A crossed-dipole antenna is often used for generating circularly polarized fields. A crossed-dipole antenna is formed from two orthogonal short-dipole antennas, one along y-axis and the other along the z-axis in the antenna's local coordinate system. This antenna object generates right-handed circularly polarized fields along the *x*-axis (defined by 0° azimuth and 0° elevation angles).

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

1. Define and set up your crossed-dipole antenna element. See "Construction" on page 1-273.
2. Call `step` to compute the antenna response according to the properties of `phased.CrossedDipoleAntennaElement`. The behavior of `step` is specific to each object in the toolbox.

Construction

`h = phased.CrossedDipoleAntennaElement` creates the system object, `h`, to model a crossed-dipole antenna element.

`h = phased.CrossedDipoleAntennaElement(Name,Value)` creates the system 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

Antenna operating frequency range

Antenna operating frequency range specified as a 1-by-2 row vector in the form of `[LowerBound HigherBound]`. This defines the frequency range over which the antenna has a response. The antenna element has no response outside the specified frequency range.

Default: `[0 1e20]`

Methods

clone	Create crossed-dipole 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
isPolarizationCapable	Polarization capability
plotResponse	Plot response pattern of antenna
release	Allow property value and input characteristics changes
step	Output response of antenna element

Examples

Plot Response of a Crossed-Dipole Antenna

Examine the response patterns of a crossed-dipole antenna used in an L-band radar with a frequency range between 1-2 GHz.

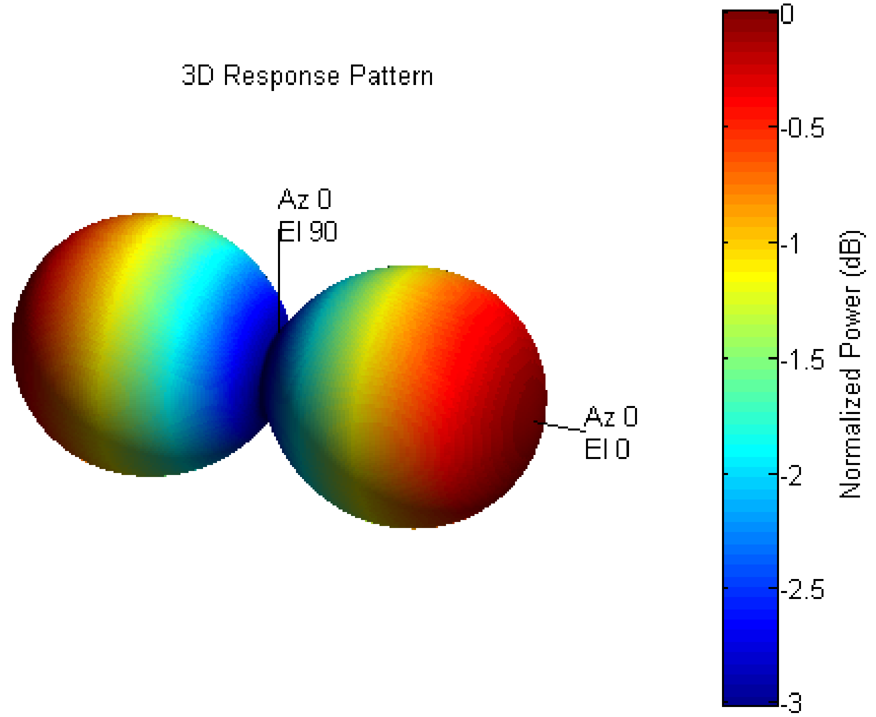
First, set up the radar parameters, and obtain the vertical and horizontal polarization responses at five different directions: elevation angles -30, -15, 0, 15 and 30 degrees, all at 0 degrees azimuth angle. The responses are computed at an operating frequency of 1.5 GHz.

```
scd = phased.CrossedDipoleAntennaElement(...
    'FrequencyRange',[1,2]*1e9);
fc = 1.5e9;
resp = step(scd,fc,[0,0,0,0,0;-30,-15,0,15,30]);
[resp.V, resp.H]
```

```
ans =
  -1.0607 + 0.0000i   0.0000 - 1.2247i
  -1.1830 + 0.0000i   0.0000 - 1.2247i
  -1.2247 + 0.0000i   0.0000 - 1.2247i
  -1.1830 + 0.0000i   0.0000 - 1.2247i
  -1.0607 + 0.0000i   0.0000 - 1.2247i
```

Next, draw a 3-D plot of the combined polarization response.

```
plotResponse(scd,fc,'Format','Polar',...
    'RespCut','3D','Polarization','C');
```



Algorithms

The total response of a crossed-dipole antenna element is a combination of its frequency response and spatial response. `phased.CrossedDipoleAntennaElement` calculates both responses using nearest neighbor interpolation, and then multiplies the responses to form the total response.

References

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

See Also

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

Purpose Create crossed-dipole 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.CrossedDipoleAntennaElement.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 `phased.CrossedDipoleAntennaElement` 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.CrossedDipoleAntennaElement.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.CrossedDipoleAntennaElement` System object supports polarization. An antenna element supports polarization if it can create or respond to polarized fields. This object supports only polarized fields.

Input Arguments

h - Crossed-dipole antenna element

Crossed-dipole antenna element specified as a `phased.CrossedDipoleAntennaElement`System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not. Because the `phased.CrossedDipoleAntennaElement` antenna element supports polarization, the returned value is always `true`.

Examples

Crossed-Dipole Antenna Element Supports Polarization

Determine whether the `phased.CrossedDipoleAntennaElement` antenna element supports polarization.

```
h = phased.CrossedDipoleAntennaElement;
isPolarizationCapable(h)
```

```
ans =

     1
```

The returned value `true (1)` shows that the crossed-dipole antenna element supports polarization.

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 System object

FREQ

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the antenna response pattern. The allowable values are `|'None' | 'Combined' | 'H' | 'V' |` where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'AzimuthAngles'

Azimuth angles for plotting element response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting element response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting element response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the U coordinates for visualizing the radiation pattern in U/V space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting element response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the V coordinates for visualizing the radiation pattern in U/V space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

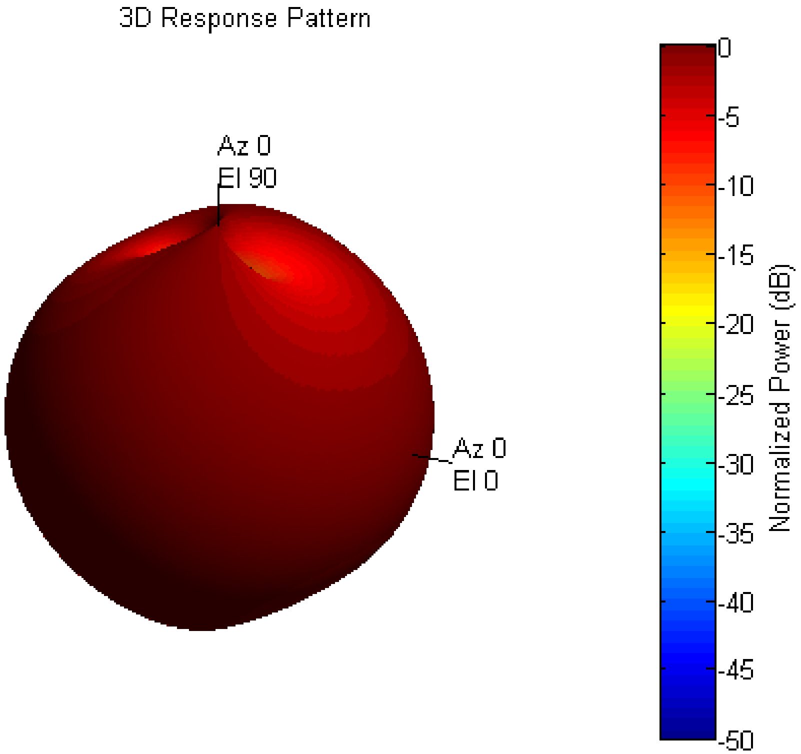
Default: `[-1:0.01:1]`

Examples

Vertical and Horizontal Responses of Crossed-Dipole Antenna

This example shows how to create a crossed-dipole antenna operating between 100 and 900 MHz and then how to plot its vertical polarization response at 250 MHz in the form of a 3-D polar plot.

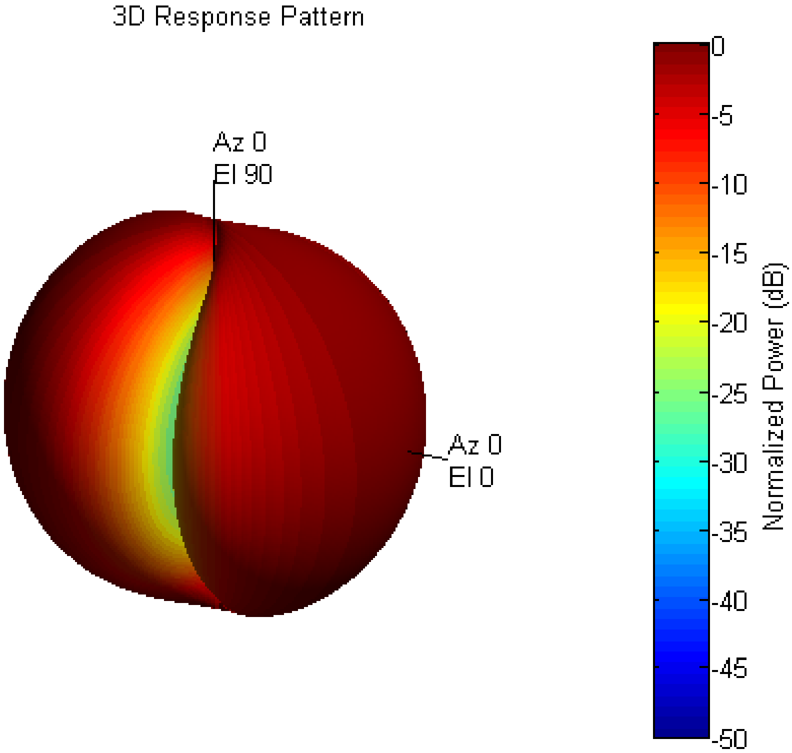
```
scd = phased.CrossedDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6);
plotResponse(scd,250e6,'Format','Polar',...
'RespCut','3D','Polarization','V');
```



The antenna pattern of the vertical-polarization component is almost isotropic and has a maximum at 0° elevation and 0° azimuth, as shown in the figure above.

Plot the antenna's horizontal polarization response. The pattern of the horizontal polarization response also has a maximum at 0° elevation and 0° azimuth but no response at $\pm 90^\circ$ azimuth.

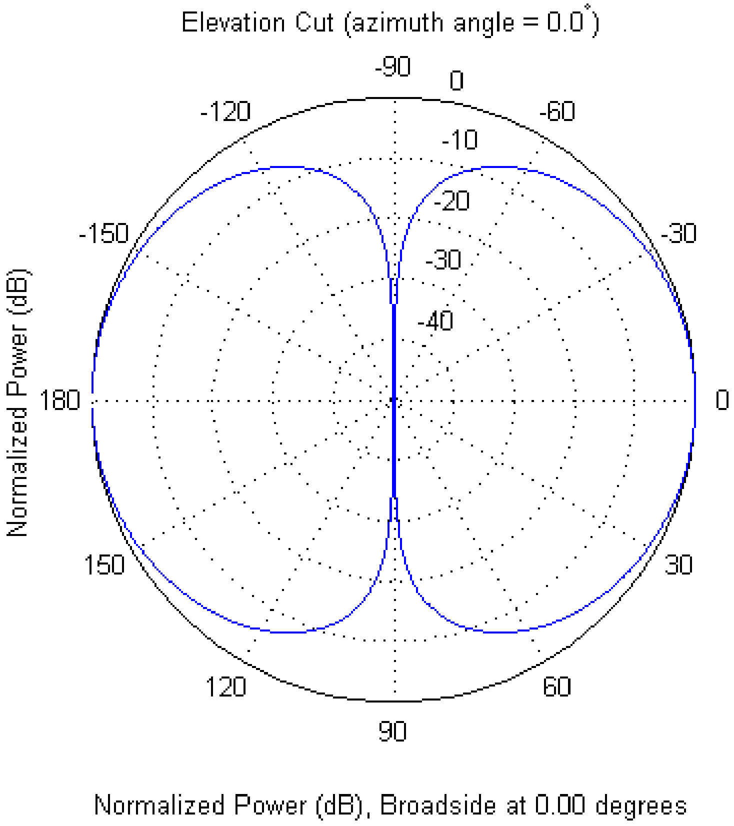
```
scd = phased.CrossedDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6);
plotResponse(scd,250e6,'Format','Polar',...
    'RespCut','3D','Polarization','H');
```



Response and Directivity of Crossed-Dipole Antenna As Elevation-Cut

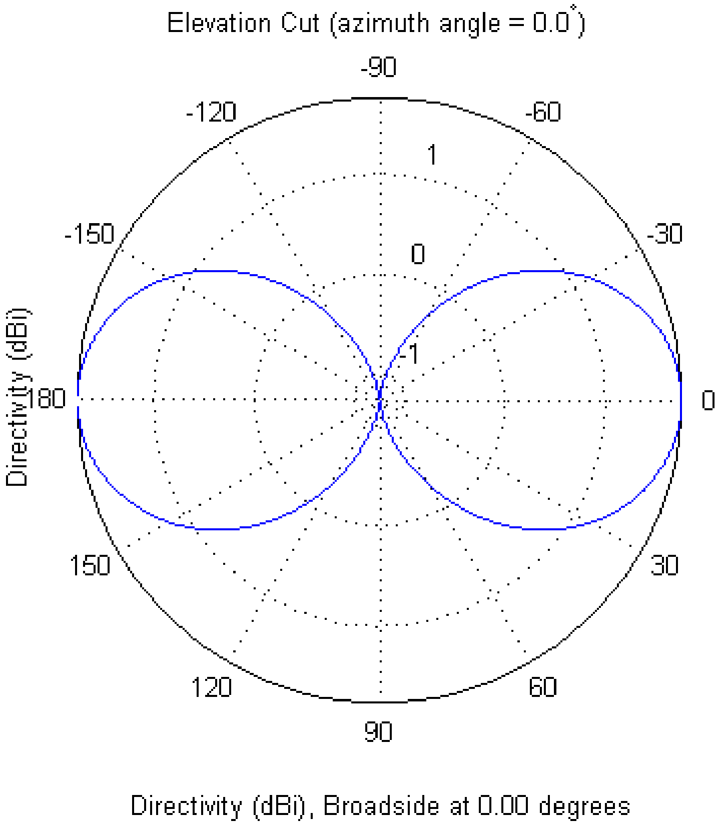
Create a crossed-dipole antenna operating between 100 and 900 MHz. Then, plot the antenna’s vertical polarization response at 250 MHz as an elevation cut. Display the response from $-90°$ to $90°$ elevation in $0.1°$ increments.

```
scd = phased.CrossedDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6);
plotResponse(scd,250e6,'Format','Polar',...
    'RespCut','El','ElevationAngles',[-90:0.1:90],...
    'Polarization','V');
```



Plot the antenna's directivity at 250 MHz as an elevation cut.

```
plotResponse(scd,250e6,'Format','Polar','Unit','dbi',...
    'RespCut','El','ElevationAngles',[-90:0.1:90]);
```



See Also `uv2azel` | `azel2uv`

phased.CrossedDipoleAntennaElement.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 the operating frequencies specified in `FREQ` and in the directions specified in `ANG`. For the crossed-dipole antenna element object, `RESP` is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`, representing the horizontal and vertical polarization components of the antenna's response. Each field is an M-by-L matrix containing the antenna response at the M angles specified in `ANG` and at the L frequencies specified in `FREQ`.

Note The object performs an initialization the first time the `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 returned as a MATLAB `struct` with fields `RESP.H` and `RESP.V`. Both `RESP.H` and `RESP.V` contain responses for the horizontal and vertical polarization components of the antenna radiation pattern. Both `RESP.H` and `RESP.V` are M-by-L matrices. In these matrices, M represents the number of angles specified in `ANG`, and L represents the number of frequencies specified in `FREQ`.

Examples

Find the response of a crossed-dipole antenna at boresight, 0° azimuth and 0° elevation, and off-boresight at 30° azimuth and 0° elevation. The antenna operates at frequencies between 100 and 900 MHz. Find the response of the antenna at these angles at 250 MHz.

```
hcd = phased.CrossedDipoleAntennaElement(...
     'FrequencyRange',[100 900]*1e6);
ang = [0 30;0 0];
fc = 250e6;
resp = step(hcd,fc,ang);
```

```
resp.H =
   0.0000 - 1.2247i
   0.0000 - 1.0607i
resp.V =
   -1.2247
   -1.2247
```

Algorithms

The total response of a crossed-dipole antenna element is a combination of its frequency response and spatial response. `phased.CrossedDipoleAntennaElement` calculates both responses using nearest neighbor interpolation, and then multiplies the responses to form the total response.

See Also

`uv2azel` | `phitheta2azel`

Purpose

Custom antenna element

Description

The `phased.CustomAntennaElement` object models an antenna element with a custom response pattern. The response pattern may be defined for polarized or non-polarized fields.

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

1. Define and set up your custom antenna element. See "Construction" on page 1-296.
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 system object, `H`, to model an antenna element with a custom response pattern. How the response pattern is specified depends upon whether polarization is desired or not. The default pattern has an isotropic spatial response.

- To create a nonpolarized response pattern, set the `SpecifyPolarizationPattern` property to `false` (default). Then, use the `RadiationPattern` property to set the response pattern.
- To create a polarized response pattern, set the `SpecifyPolarizationPattern` property to `true`. Then, use any or all of the `HorizontalMagnitudePattern`, `HorizontalPhasePattern`, `VerticalMagnitudePattern`, and `VerticalPhasePattern` properties to set the response pattern.

The output response of the `step` method depends on whether polarization is set or not.

`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

Response and pattern frequency vector

Specify the frequencies (in Hz) at which the frequency response and antenna patterns are to be returned, as a 1-by-L row vector. The elements of the vector must be in increasing order. The antenna element has no response outside the frequency range specified by the minimum and maximum elements of the frequency vector.

Default: `[0 1e20]`

AzimuthAngles

Azimuth angles

Specify the azimuth angles (in degrees) as a length-P vector. These values are the azimuth angles where the custom radiation pattern is to be specified. P must be greater than 2. The azimuth angles should lie between –180 and 180 degrees and be in strictly increasing order.

Default: `[-180:180]`

ElevationAngles

Elevation angles

Specify the elevation angles (in degrees) as a length-Q vector. These values are the elevation angles where the custom radiation pattern is to be specified. Q must be greater than 2. The elevation angles should lie between –90 and 90 degrees and be in strictly increasing order.

Default: `[-90:90]`

FrequencyResponse

Frequency responses of antenna element

Specify the frequency responses in decibels measured at the frequencies defined in `FrequencyVector` property as a 1-by-L row vector where L must equal the length of the vector specified in the `FrequencyVector` property.

Default: `[0 0]`

SpecifyPolarizationPattern

Polarized array response

- When the `SpecifyPolarizationPattern` property is set to `false`, nonpolarized radiation is transmitted or received by the antenna element. In this case, use the `RadiationPattern` property to set the antenna response pattern.
- When the `SpecifyPolarizationPattern` property is set to `true`, polarized radiation is transmitted or received by the antenna element. In this case, use the `HorizontalMagnitudePattern` and `HorizontalPhasePattern` properties to set the horizontal polarization response pattern and the `VerticalMagnitudePattern` and `VerticalPhasePattern` properties to set the vertical polarization response pattern.

Default: `false`

RadiationPattern

Magnitude of combined antenna radiation pattern

The magnitude of the combined polarization antenna radiation pattern specified as a Q-by-P matrix or a Q-by-P-by-L array. This property is used only when the `SpecifyPolarizationPattern` property is set to `false`. Magnitude units are in dB.

- If the value of this property is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.

- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

If the pattern contains a `NaN` at any azimuth and elevation direction, it is converted to `-Inf`, indicating zero response in that direction. The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern should contain azimuth angles in the range`[ 180,180]` degrees. You should also set the range of elevation angles to `[ 90,90]` degrees.

Default: A 181-by-361 matrix with all elements equal to 0 dB

HorizontalMagnitudePattern

Magnitude of horizontal polarization component of antenna radiation pattern

The magnitude of the horizontal polarization component of the antenna radiation pattern specified as a Q-by-P matrix or a Q-by-P-by-L array. This property is used only when the `SpecifyPolarizationPattern` property is set to `true`. Magnitude units are in dB.

- If the value of this property is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

If the magnitude pattern contains a `NaN` at any azimuth and elevation direction, it is converted to `-Inf`, indicating zero response in that direction. The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern should contain azimuth angles in the range `[ 180,180]` nd elevation angles in the range `[ 90,90]` .

Default: A 181-by-361 matrix with all elements equal to 0 dB

HorizontalPhasePattern

Phase of horizontal polarization component of antenna radiation pattern

The phase of the horizontal polarization component of the antenna radiation pattern specified as a Q-by-P matrix or a Q-by-P-by-L array. This property is used only when the `SpecifyPolarizationPattern` property is set to `true`. Phase units are in degrees.

- If the value of this property is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern should contain azimuth angles in the range`[ 180,180]` and elevation angles in the range `[ 90,90]` .

Default: A 181-by-361 matrix with all elements equal to 0°

VerticalMagnitudePattern

Magnitude of vertical polarization component of antenna radiation pattern

The magnitude of the vertical polarization component of the antenna radiation pattern specified as a Q-by-P matrix or a Q-by-P-by-L array. This property is used only when the `SpecifyPolarizationPattern` property is set to `true`. Magnitude units are in dB.

- If the value of this property is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

If the pattern contains a `NaN` at any azimuth and elevation direction, it is converted to `-Inf`, indicating zero response in that direction. The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern should contain azimuth angles in the range`[ 180,180]` and elevation angles in the range `[ 90,90]` .

Default: A 181-by-361 matrix with all elements equal to 0 dB

VerticalPhasePattern

Phase of vertical polarization component of antenna radiation pattern

The phase of the vertical polarization component of the antenna radiation pattern specified as a Q-by-P matrix or a Q-by-P-by-L array. This property is used only when the `SpecifyPolarizationPattern` property is set to `true`. Phase units are in degrees.

- If the value of this property is a Q-by-P matrix, the same pattern is applied to *all* frequencies specified in the `FrequencyVector` property.
- If the value is a Q-by-P-by-L array, each Q-by-P page of the array specifies a pattern for the *corresponding* frequency specified in the `FrequencyVector` property.

The custom antenna object uses interpolation to estimate the response of the antenna at a given direction. To avoid interpolation errors, the custom response pattern should contain azimuth angles in the range`[ 180,180]` and elevation angles in the range `[ 90,90]` .

Default: A 181-by-361 matrix with all elements equal to 0°

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
isPolarizationCapable	Polarization capability
plotResponse	Plot response pattern of antenna
release	Allow property value and input characteristics changes
step	Output response of antenna element

Examples

Response and Directivity of Custom Antenna

Create a user-defined antenna with cosine pattern, and plot an elevation cut of the antenna's power response.

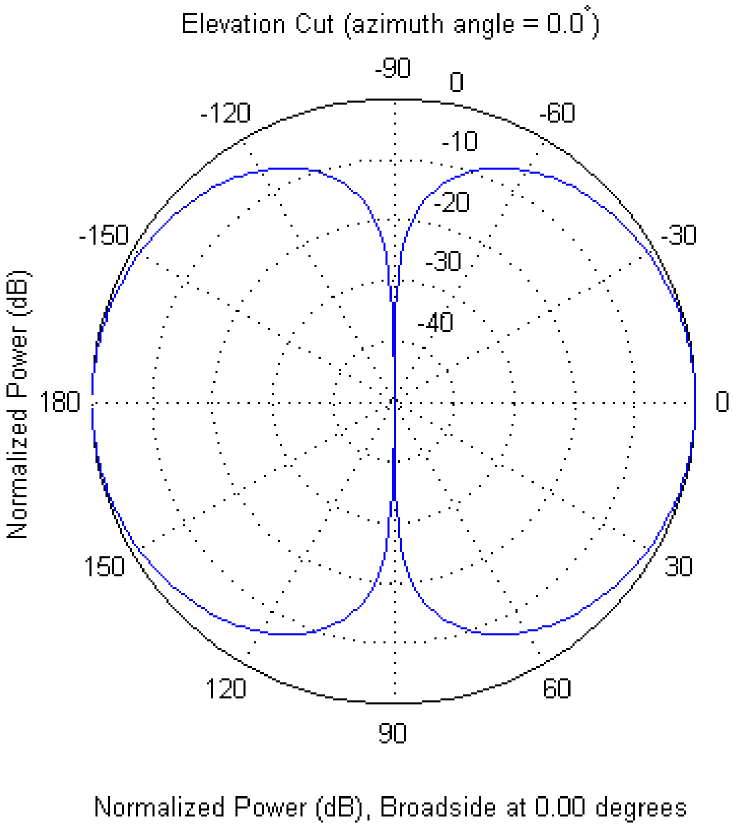
The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna operates at 1 GHz. Get the response at 0 degrees azimuth and 30 degrees elevation.

```
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;30])
```

```
resp =

    0.8660
```

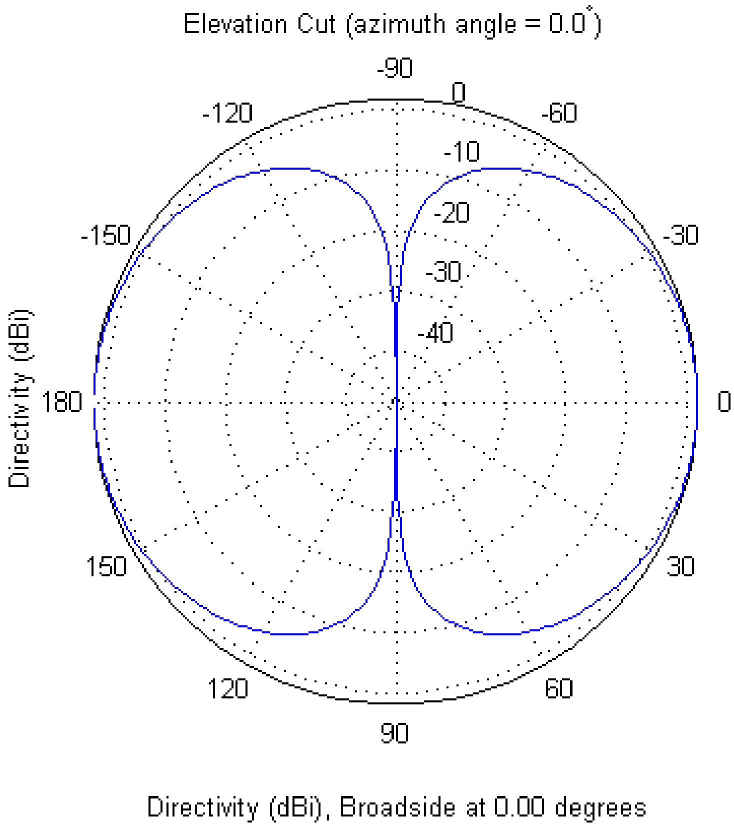
Plot an elevation cut of the power response.

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



Plot an elevation cut of the directivity.

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



Antenna Radiation Pattern in U-V Coordinates

Define a custom antenna in $u-v$ space. Then, calculate and plot the response.

Define the radiation pattern (in dB) of an antenna in terms of u and v coordinates within the unit circle.

```
u = -1:0.01:1;
v = -1:0.01:1;
```

```
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Create an antenna with this radiation pattern. Convert $u-v$ coordinates to azimuth and elevation coordinates.

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

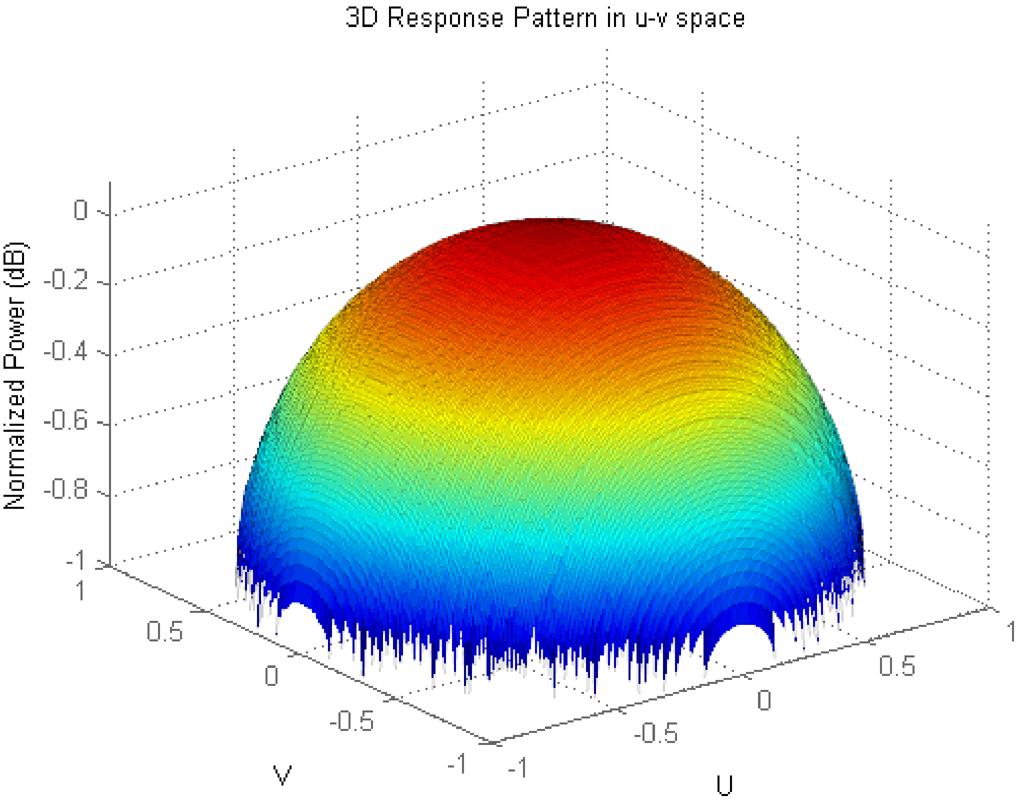
Calculate the response in the direction $u=0.5$, $v=0$. Assume the antenna operates at 1 GHz. The output of the step method is in linear units.

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

```
resp =

    1.1048
```

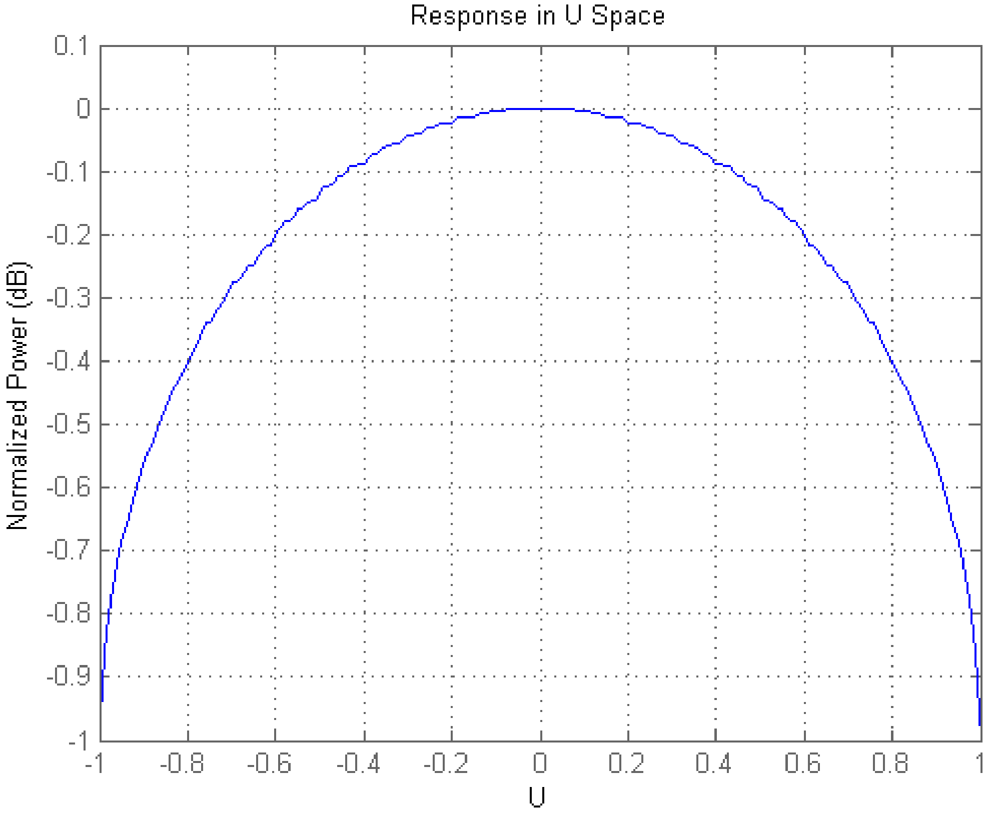
Plot the 3D response in UV coordinates.

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



Display the antenna response as a line plot in UV coordinates.

```
plotResponse(ha,fc,'Format','UV');
```



Polarized Antenna Radiation Patterns

Model a short dipole antenna oriented along the x -axis of the local antenna coordinate system. For this type of antenna, the horizontal and vertical components of the electric field are given by $E_H = \frac{j\omega\mu IL}{4\pi r}\sin(\mathrm{az})$ and $E_V = -\frac{j\omega\mu IL}{4\pi r}\sin(\mathrm{el})\cos(\mathrm{az})$.

Specify a normalized radiation pattern of a short dipole antenna terms of azimuth, az , and elevation, el , coordinates. The vertical and horizontal radiation patterns are normalized to a maximum of unity.

```
az = [-180:180];
el = [-90:90];
[az_grid,el_grid] = meshgrid(az,el);
horz_pat_azel = ...
    mag2db(abs(sind(az_grid)));
vert_pat_azel = ...
    mag2db(abs(sind(el_grid).*cosd(az_grid)));
```

Set up the antenna. Specify the `SpecifyPolarizationPattern` property to produce polarized radiation. In addition, set the `HorizontalMagnitudePattern` and `VerticalMagnitudePattern` properties. The `HorizontalPhasePattern` and `VerticalPhasePattern` properties take default values of zero.

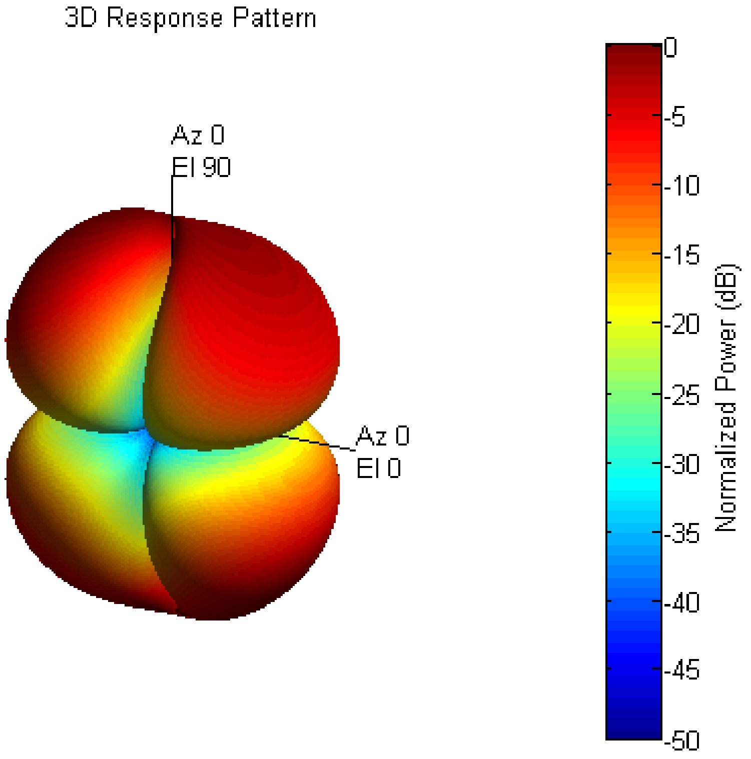
```
ha = phased.CustomAntennaElement(...
    'AzimuthAngles',az,'ElevationAngles',el,...
    'SpecifyPolarizationPattern',true,...
    'HorizontalMagnitudePattern',horz_pat_azel,...
    'VerticalMagnitudePattern',vert_pat_azel);
```

Assume the antenna operates at 1 GHz.

```
fc = 1e9;
```

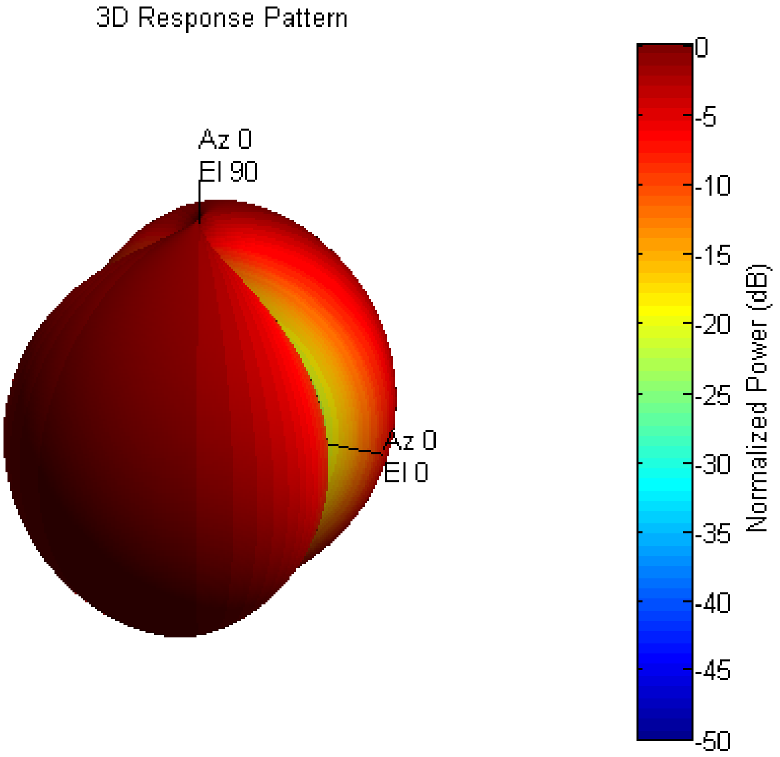
Display the vertical response pattern.

```
plotResponse(ha,fc,'Format','Polar',...
    'RespCut','3D','Polarization','V');
```



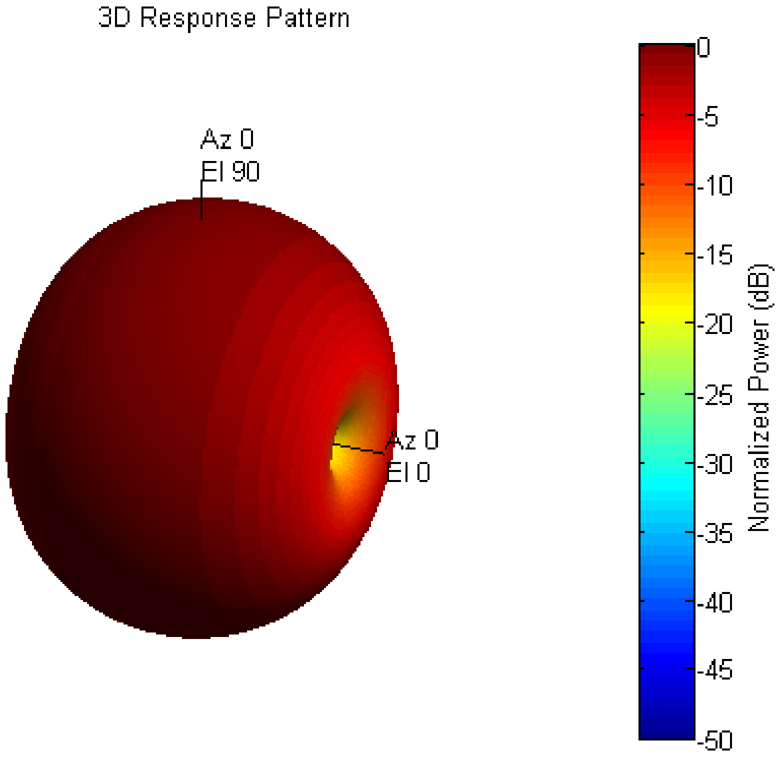
Display the horizontal response pattern.

```
plotResponse(ha,fc,'Format','Polar',...
    'RespCut','3D','Polarization','H');
```



The combined polarization response, shown below, best illustrates the x -axis polarity of the dipole.

```
plotResponse(ha,fc,'Format','Polar',...
    'RespCut','3D','Polarization','C');
```



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.CrossedDipoleAntennaElement` | `phased.CosineAntennaElement` | `phased.IsotropicAntennaElement`

| `phased.ShortDipoleAntennaElement` | `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.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.CustomAntennaElement` System object supports polarization. An antenna element supports polarization if it can create or respond to polarized fields. This antenna object supports both polarized and nonpolarized fields.

Input Arguments

h - Custom antenna element

Custom antenna element specified as a `phased.CustomAntennaElement`.

Output Arguments

flag - Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not. The returned value depends upon the value of the `SpecifyPolarizationPattern` property. If `SpecifyPolarizationPattern` is `true`, then `flag` is `true`. Otherwise it is `false`.

Examples

Custom Antenna Element Polarization Capability

Determine whether the `phased.CustomAntennaElement` antenna element supports polarization when `SpecifyPolarizationPattern` is set to `true`.

```
h = phased.CustomAntennaElement(...
    'SpecifyPolarizationPattern',true);
isPolarizationCapable(h)
```

```
ans =

     1
```

The returned value `true (1)` shows that this antenna element supports polarization when the `'SpecifyPolarizationPattern'` property is set to `true`.

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 System object

`FREQ`

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the antenna response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'AzimuthAngles'

Azimuth angles for plotting element response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting element response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting element response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting element response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Plot Response and Directivity of Custom Antenna

Create a custom antenna with a cosine pattern. Then, plot the antenna's response.

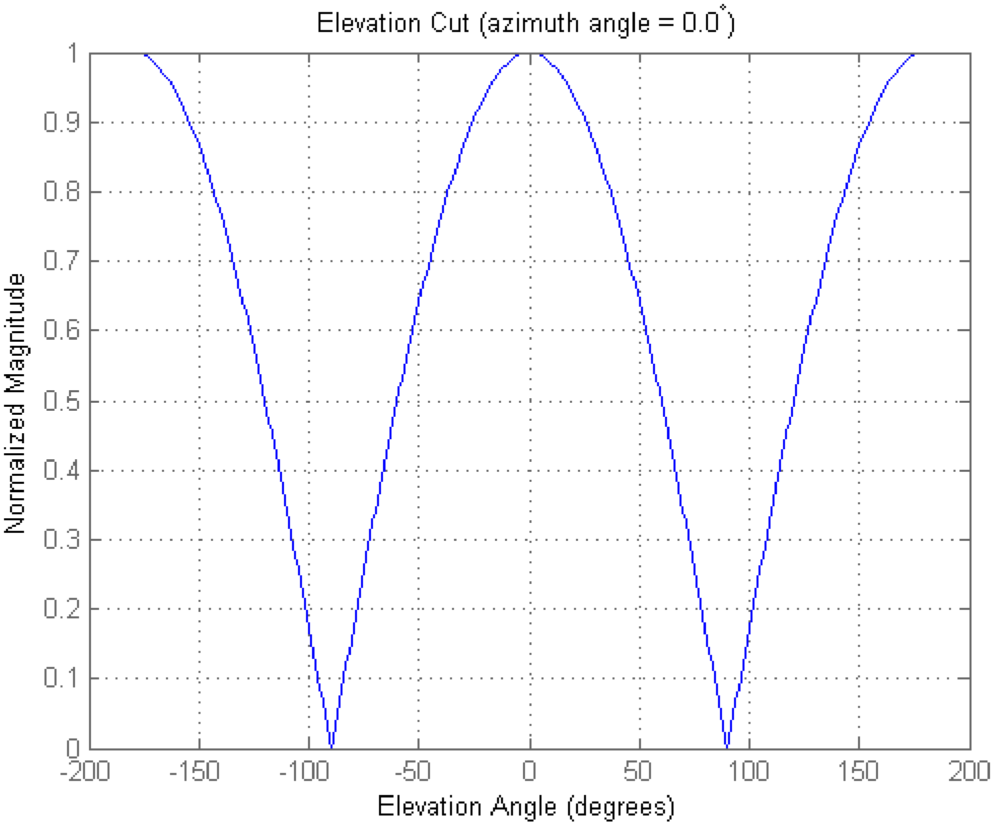
Create the antenna and calculate the response. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna works at 1 GHz.

```
fc = 1e9;
sCust = phased.CustomAntennaElement;
```

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

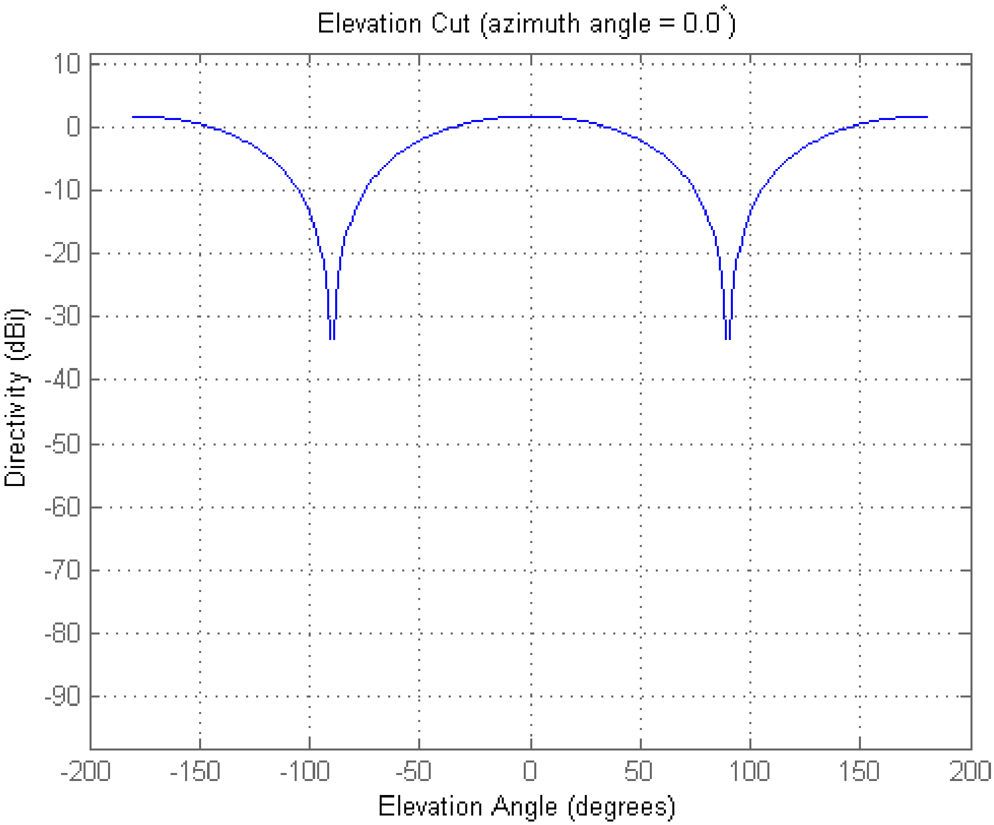
Plot an elevation cut of the magnitude response as a line plot.

```
plotResponse(sCust,fc,'RespCut','El','Format','Line','Unit','mag');
```



Plot an elevation cut of the directivity as a line plot, showing that the maximum directivity is approximately 2 dB.

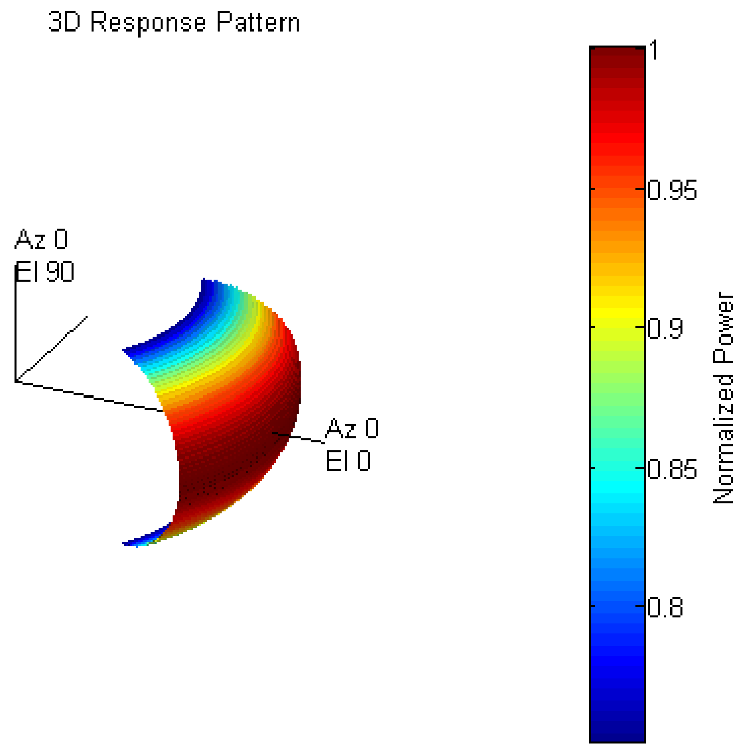
```
plotResponse(sCust,fc,'RespCut','El','Format','Line','Unit','dbi');
```



Plot Response of Custom Antenna Over Selected Range of Angles

Create an antenna with a custom response. The user-defined pattern is omnidirectional in the azimuth direction and has a cosine pattern in the elevation direction. Assume the antenna operates at a frequency of 1 GHz. Display the 3-D response for a 60 degree range of azimuth and elevation angles centered at 0 degrees azimuth and 0 degrees elevation in 0.1 degree increments.

```
fc = 1e9;
sCust = phased.CustomAntennaElement;
sCust.AzimuthAngles = -180:180;
sCust.ElevationAngles = -90:90;
sCust.RadiationPattern = mag2db(repmat(cosd(sCust.ElevationAngles)',..
    1,numel(sCust.AzimuthAngles)));
resp = step(sCust,fc,[0;0]);
plotResponse(sCust,fc,'RespCut','3D','Format','Polar',...
    'AzimuthAngles',[-30:0.1:30],'ElevationAngles',...
    [-30:0.1:30],'Unit','pow');
```



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`. The form of `RESP` depends upon whether the antenna element supports polarization as determined by the `SpecifyPolarizationPattern` property. If `SpecifyPolarizationPattern` is set to `false`, `RESP` is an M-by-L matrix containing the antenna response at the M angles specified in `ANG` and at the$L$ frequencies specified in `FREQ`. If `SpecifyPolarizationPattern` is set to `true`, `RESP` is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`, representing the antenna's response in horizontal and vertical polarization, respectively. Each field is an M-by-L matrix containing the antenna response at the M angles specified in `ANG` and at the$L$ frequencies specified in `FREQ`.

Note The object performs an initialization the first time the `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. The output depends on whether the antenna element supports polarization or not.

- If the antenna element does not support polarization, `RESP` is an M-by-L matrix. In this matrix, M represents the number of angles specified in `ANG` while L represents the number of frequencies specified in `FREQ`.
- If the antenna element supports polarization, `RESP` is a MATLAB `struct` with fields `RESP.H` and `RESP.V` containing responses for the horizontal and vertical polarization components of the antenna radiation pattern. `RESP.H` and `RESP.V` are M-by-L matrices. In these matrices, M represents the number of angles specified in `ANG` while L represents the number of frequencies specified in `FREQ`.

Examples

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]);

resp =

     1
```

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 1-333.
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: `[0 1e20]`

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

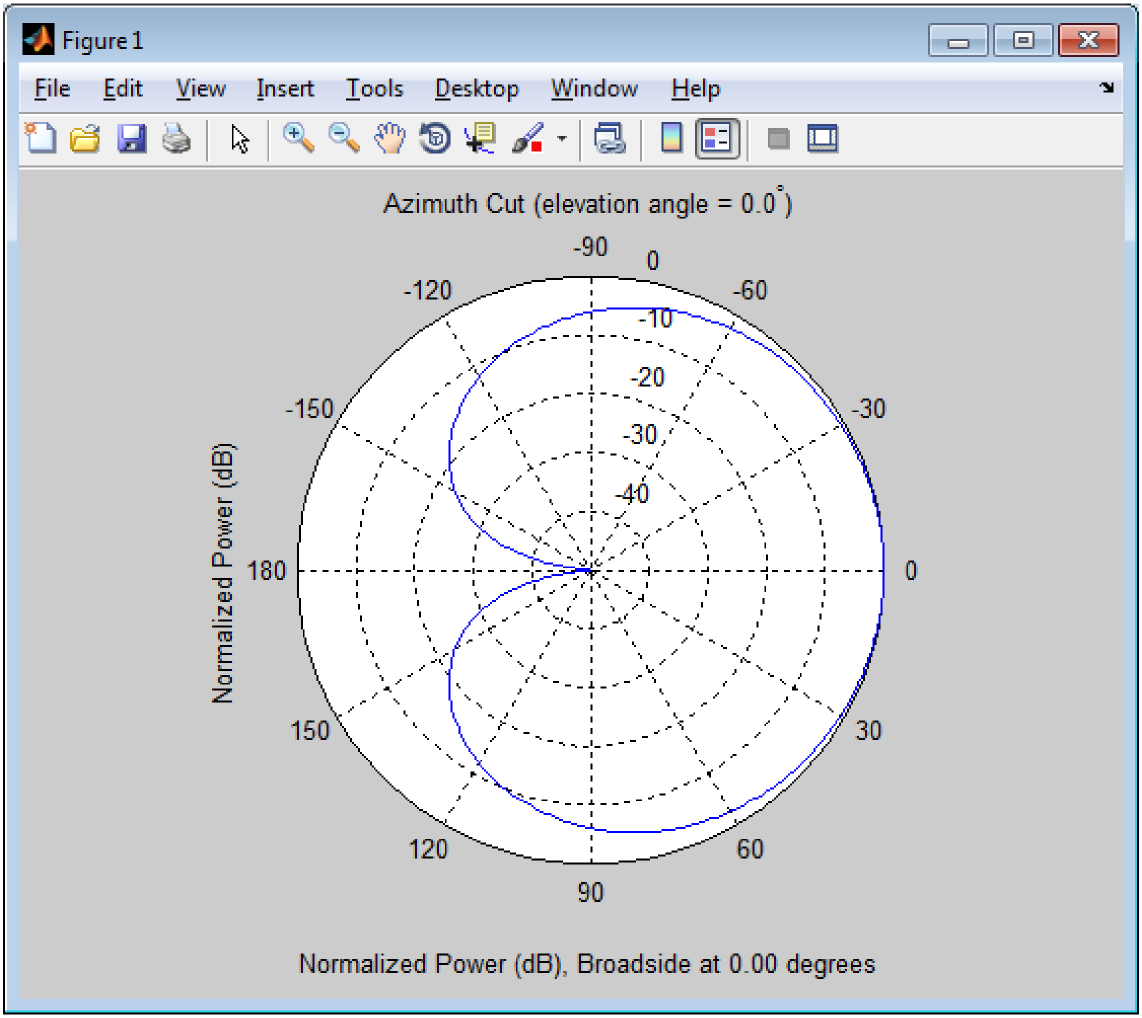
Methods

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
isPolarizationCapable	Polarization capability
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`.

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.

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.isPolarizationCapable

Purpose

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.CustomMicrophoneElement` supports polarization. An element supports polarization if it can create or respond to polarized fields. This microphone element, as with all microphone elements, does not support polarization.

Input Arguments

h - Custom microphone element

Custom microphone element specified as a `phased.CustomMicrophoneElement` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the microphone element supports polarization or `false` if it does not. Because the `phased.CustomMicrophoneElement` object does not support polarization, `flag` is always returned as `false`.

Examples

Custom Microphone Element does not Support Polarization

Show that the `phased.CustomMicrophoneElement` microphone element does not support polarization.

```
h = phased.CustomMicrophoneElement;
isPolarizationCapable(h)
```

```
ans =

     0
```

The returned value `false (0)` shows that the custom microphone element does not support polarization.

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 System object

FREQ

Operating frequency in Hertz specified as a scalar or 1–by-K row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the antenna response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'AzimuthAngles'

Azimuth angles for plotting element response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting element response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting element response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting element response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

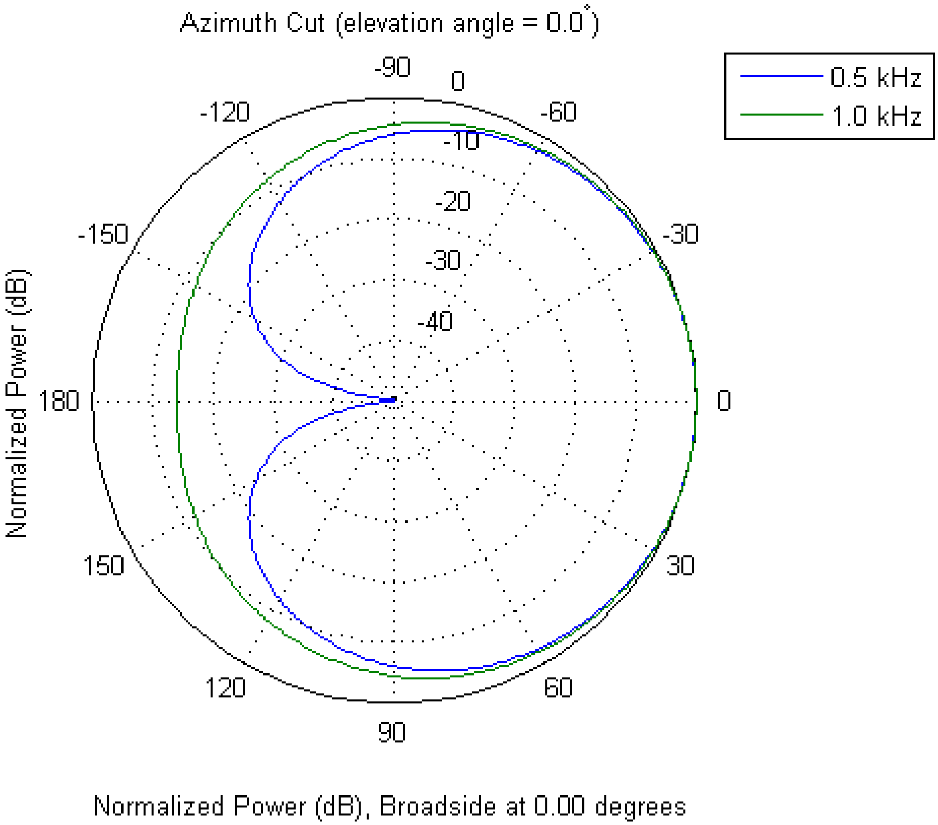
Azimuth Response and Directivity of Cardioid Microphone

Design a cardioid microphone to operate in the frequency range between 500 and 1000 Hz.

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

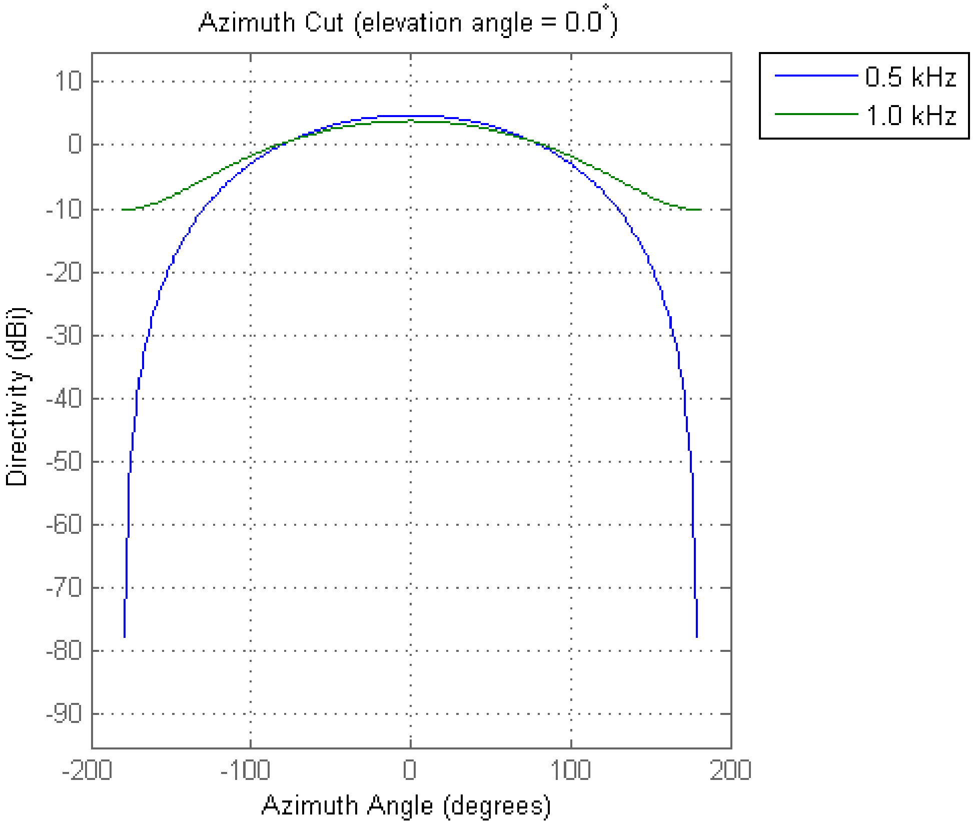
Display a polar plot of an azimuth cut of the response at 500 Hz and 1000 Hz.

```
fc = 500;
plotResponse(h,[fc 2*fc],'RespCut','Az','Format','Polar');
```



Plot the directivity as a line plot for the same two frequencies.

```
plotResponse(h,[fc 2*fc],'RespCut','Az','Format','Line','Unit','dbi');
```



Response of Cardioid Microphone in U/V Space

Plot a u -cut of the response of a custom cardioid microphone that is designed to operate in the frequency range 500-1000 Hz.

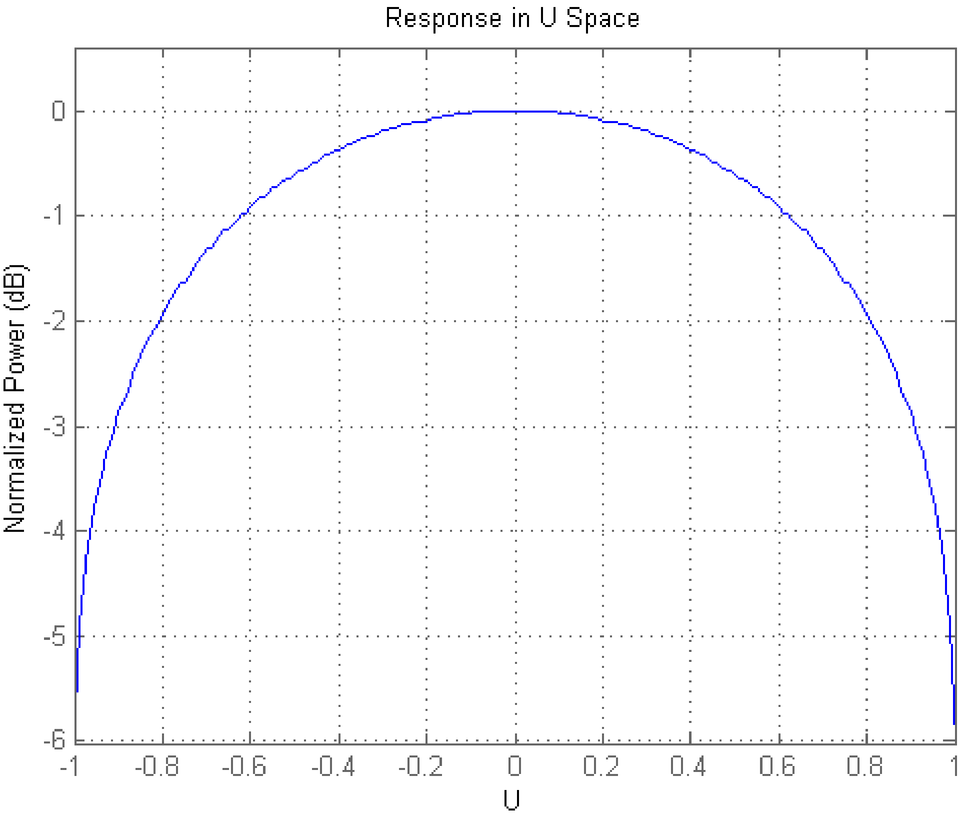
Create a cardioid microphone.

```
h = phased.CustomMicrophoneElement;
h.PolarPatternFrequencies = [500 1000];
```

```
h.PolarPattern = mag2db([...
    0.5+0.5*cosd(h.PolarPatternAngles);...
    0.6+0.4*cosd(h.PolarPatternAngles)]);
```

Plot the response.

```
fc = 500;
plotResponse(h,fc,'Format','UV');
```



3-D Response of Cardioid Microphone Over Restricted Range of Angles

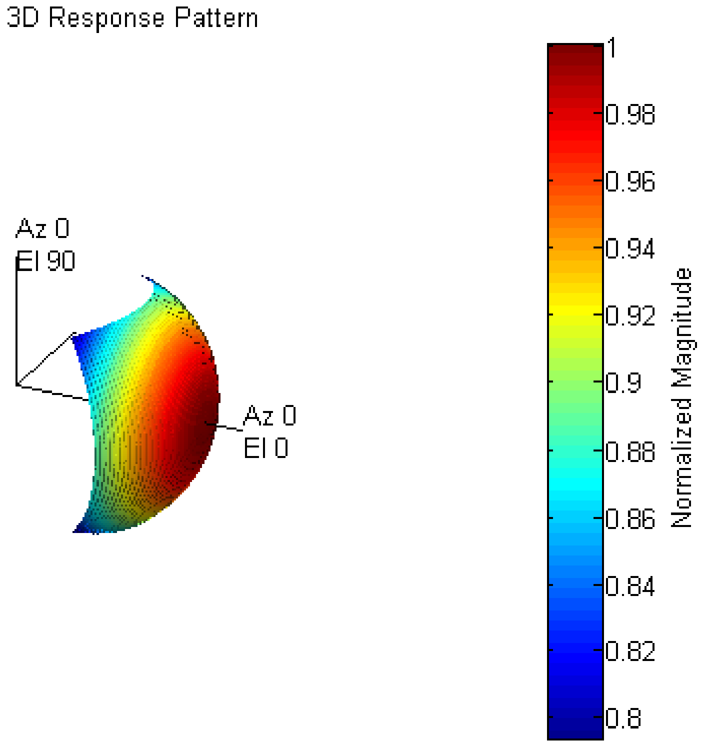
Plot the 3-D response of a custom cardioid microphone in space but with both the azimuth and elevation angles restricted to the range -40 to 40 degrees in 0.1 degree increments.

Create a custom microphone element with a cardioid pattern.

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

Plot the 3-D response.

```
fc = 500;
plotResponse(h,fc,'Format','polar','RespCut','3D',...
    'Unit','mag','AzimuthAngles',[-40:0.1:40],...
    'ElevationAngles',[-40:0.1:40]);
```



See Also `uv2azel` | `azel2uv`

phased.CustomMicrophoneElement.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.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`

phased.DPCACanceller

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 1-356.
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);
```

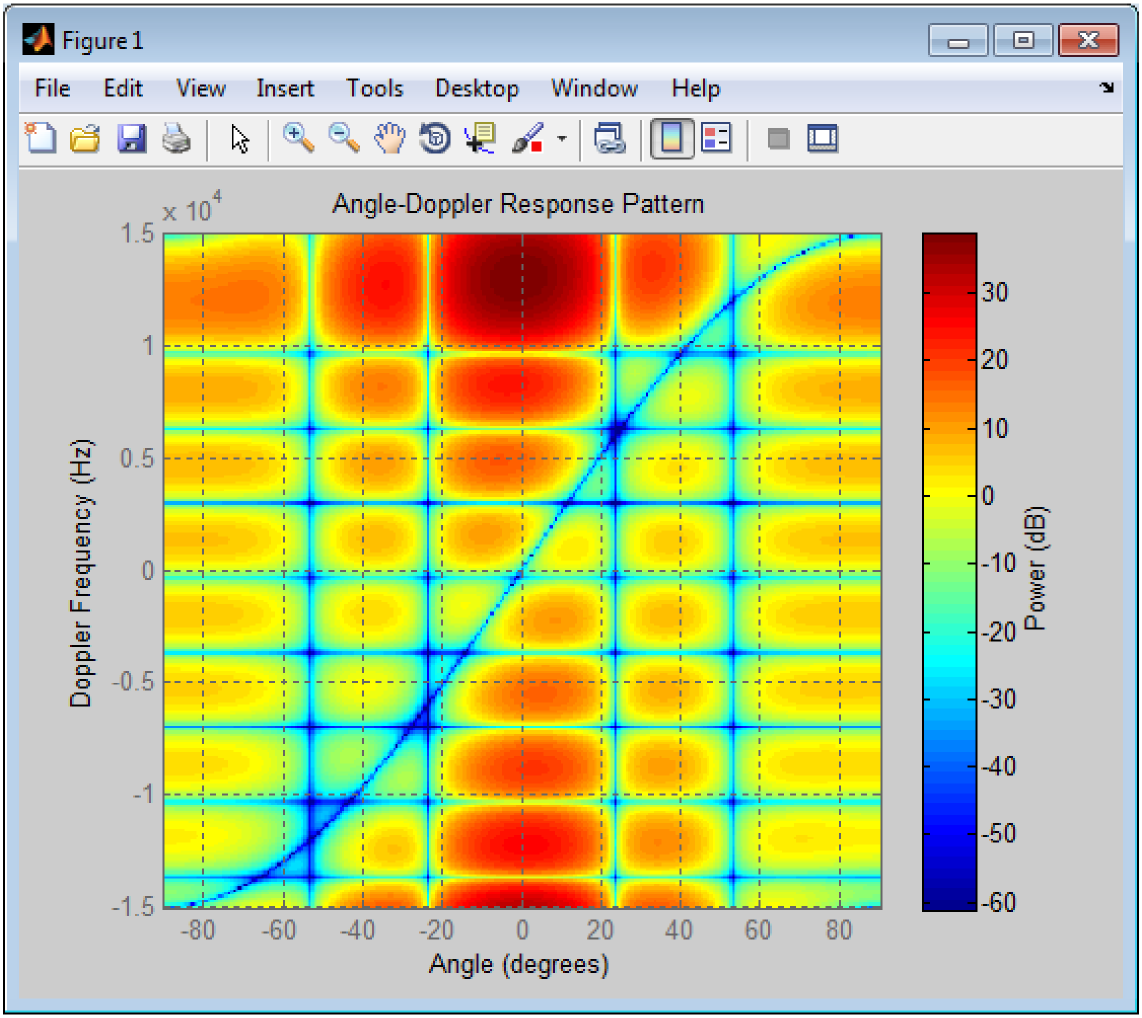


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Angle-Doppler Response Pattern
x 10^4
1.5
1
0.5
0
-0.5
-1
-1.5
Doppler Frequency (Hz)
-80
-60
-40
-20
0
20
40
60
80
Angle (degrees)
30
20
10
0
-10
-20
-30
-40
-50
-60
Power (dB)

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`

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`.

phased.DPCACanceller.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.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.

phased.DPCACanceller.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 `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.

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.DPCACanceller.step

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`

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 1-371.
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`.

phased.ElementDelay.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.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.

phased.ElementDelay.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 `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.

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 1-380.
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`

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.

phased.ESPRITEstimator.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 `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.

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.ESPRITEstimator.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)));
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 1-391.
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`

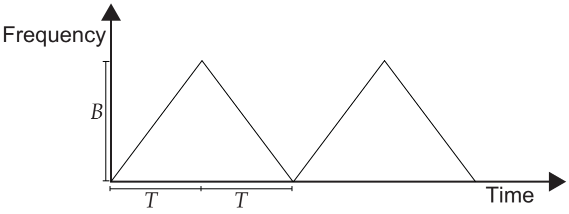
Methods

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

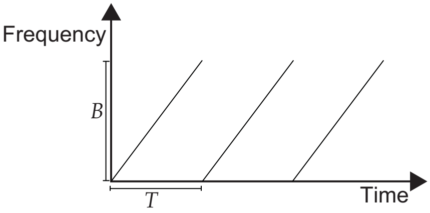
Triangle Sweep

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 2*T*.


Frequency
B
T
T
Time

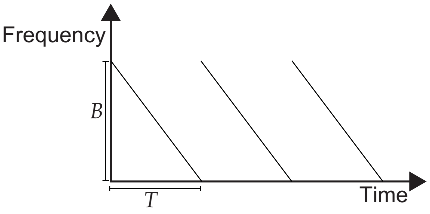
Upsweep

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.


Frequency
B
T
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.


Frequency
B
T
Time

Examples

FMCW Waveform Plot

Create and plot an upsweep FMCW waveform.

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

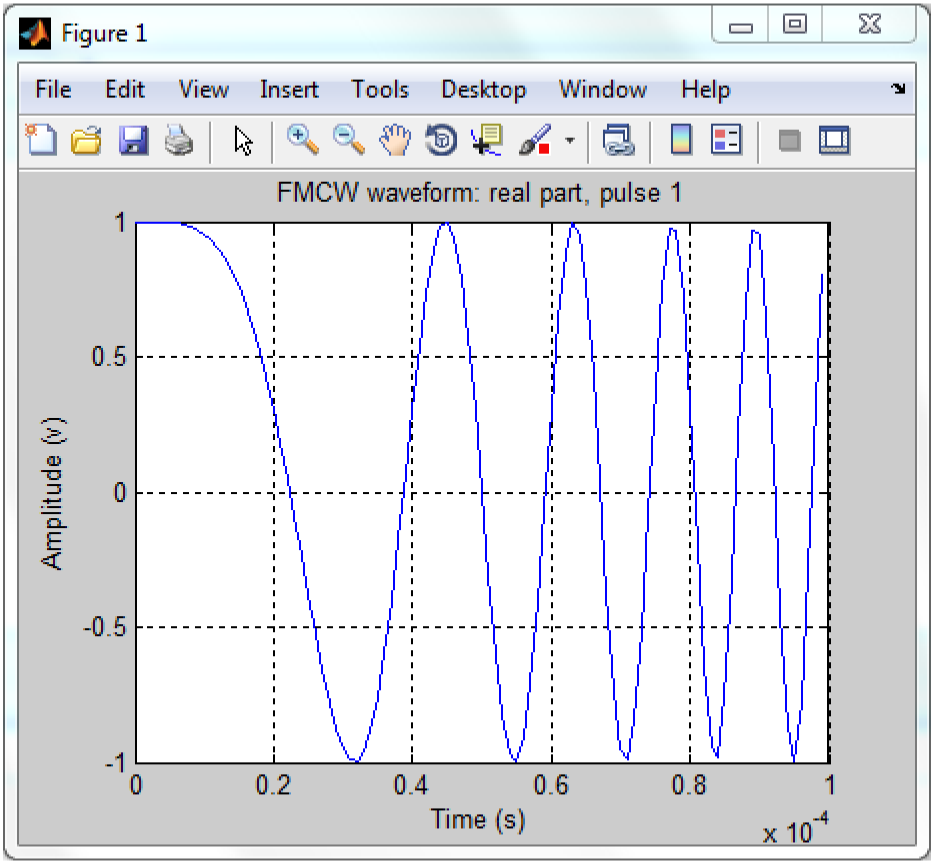


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
FMCW waveform: real part, pulse 1
Amplitude (v)
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1
Time (s)
x 10^-4

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');
```

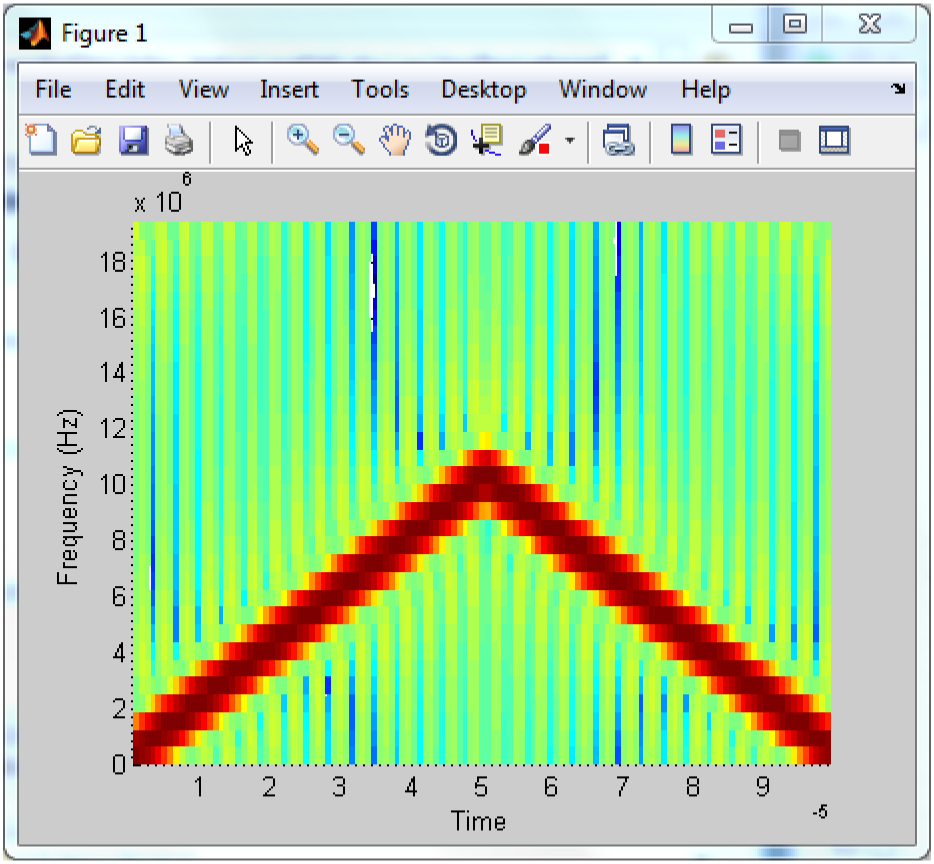


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
x 10^6
0
2
4
6
8
10
12
14
16
18
Frequency (Hz)
1
2
3
4
5
6
7
8
9
-5
Time

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` | `range2bw` | `phased.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.

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.FMCWWaveform.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 `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 `PlotType` 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. `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);
```

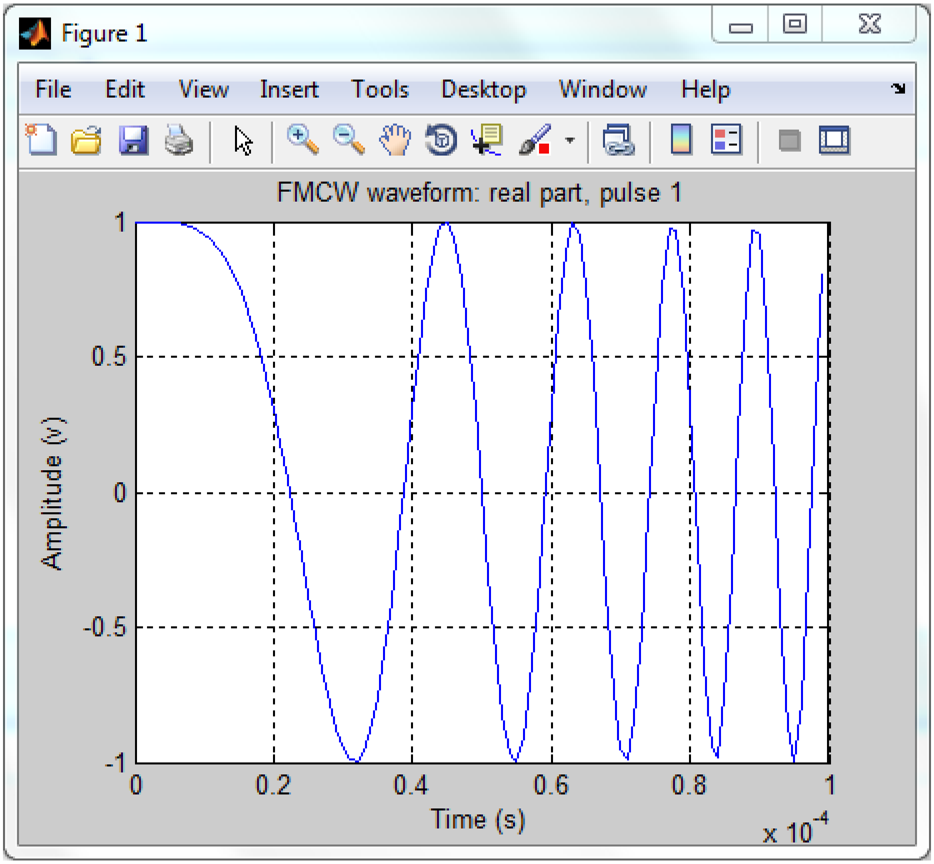


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
FMCW waveform: real part, pulse 1
Amplitude (v)
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1
Time (s)
x 10^-4

phased.FMCWWaveform.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.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.

phased.FMCWWaveform.step

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');
```

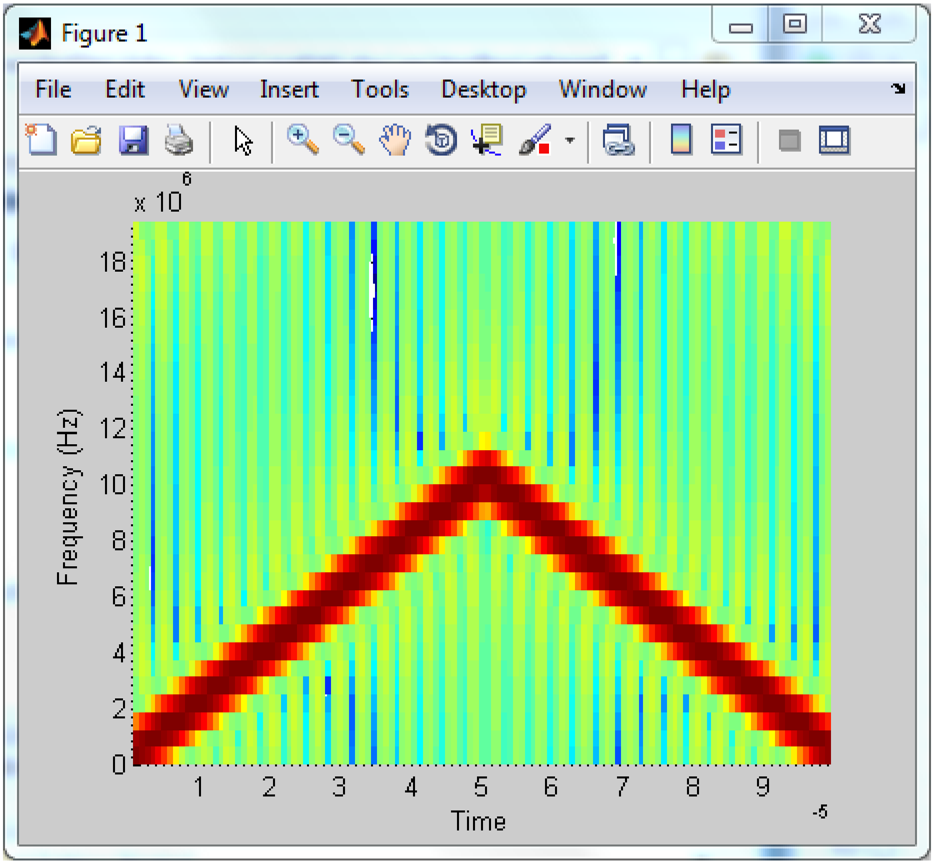


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
x 10^6
18
16
14
12
10
8
6
4
2
0
Frequency (Hz)
1
2
3
4
5
6
7
8
9
-5
Time

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 1-410.
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 $Y(t)=x(t-\text{tau})/L$. 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

`fspl` | `phased.RadarTarget`

phased.FreeSpace.clone

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.

phased.FreeSpace.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 `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.

phased.FreeSpace.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 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.

The form of `X` depends upon whether polarization is simulated or not. If polarization is not simulated, `X` is a column vector.

If polarization is simulated `X` is a MATLAB `struct` containing two alternate ways of representing the polarized signal:

- `X.X`, `X.Y`, and `X.Z` representing the *x*, *y*, and *z* components of the polarized signal.

- `X.H` and `X.V` representing the horizontal and vertical components of the polarized signal.

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 or MATLAB `struct`, depending upon the form of the input argument `X`. If `X` is a column vector, `Y` is also a column vector with same dimensions. If `X` is a `struct`, `Y` is also a `struct` with the same fields. Each field in `Y` contains the resulting signal of the corresponding field in `X`. The output `Y` is the signal arriving at the propagation destination within the current time frame, which is the time occupied by the current input. Whenever it takes longer than the current time frame for the signal to propagate from the origin to the destination, the output contains no contribution from the input of the current time frame.

Examples

Signal Propagation from Stationary Radar to Stationary Target

Calculate the 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 $Y(t)=x(t-\text{tau})/L$. 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.

phased.FrostBeamformer

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 1-424.
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');
```

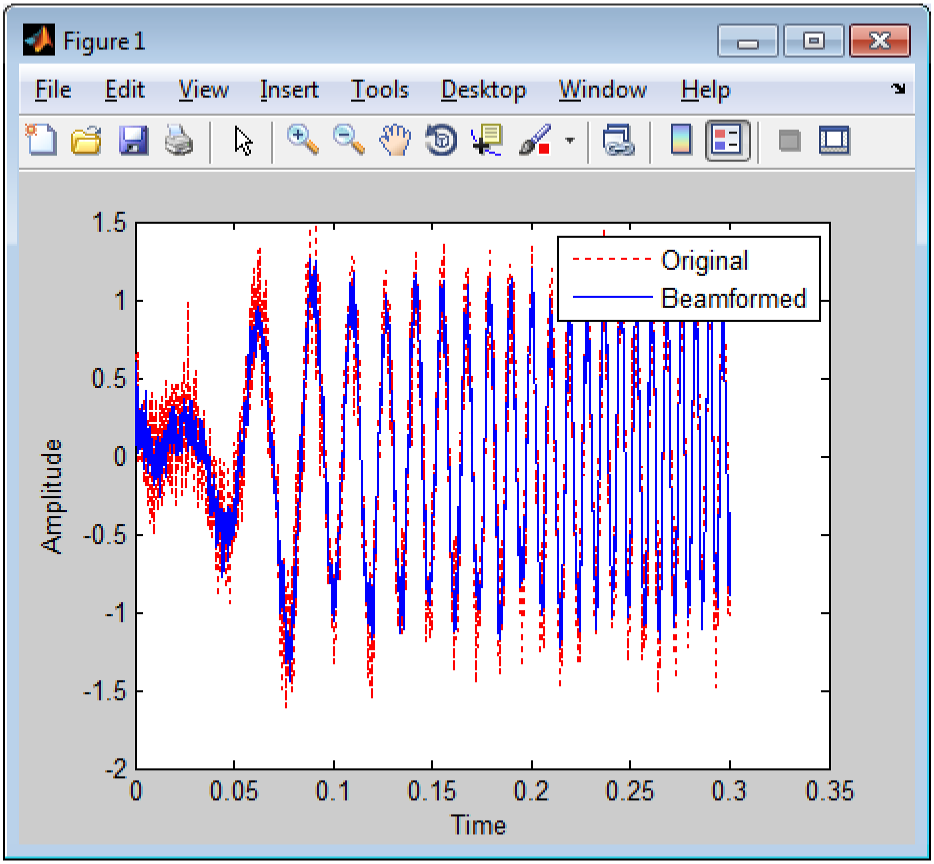


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Original
Beamformed
1.5
1
0.5
0
-0.5
-1
-1.5
-2
Amplitude
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
Time

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 1-439.
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 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'`	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);
```

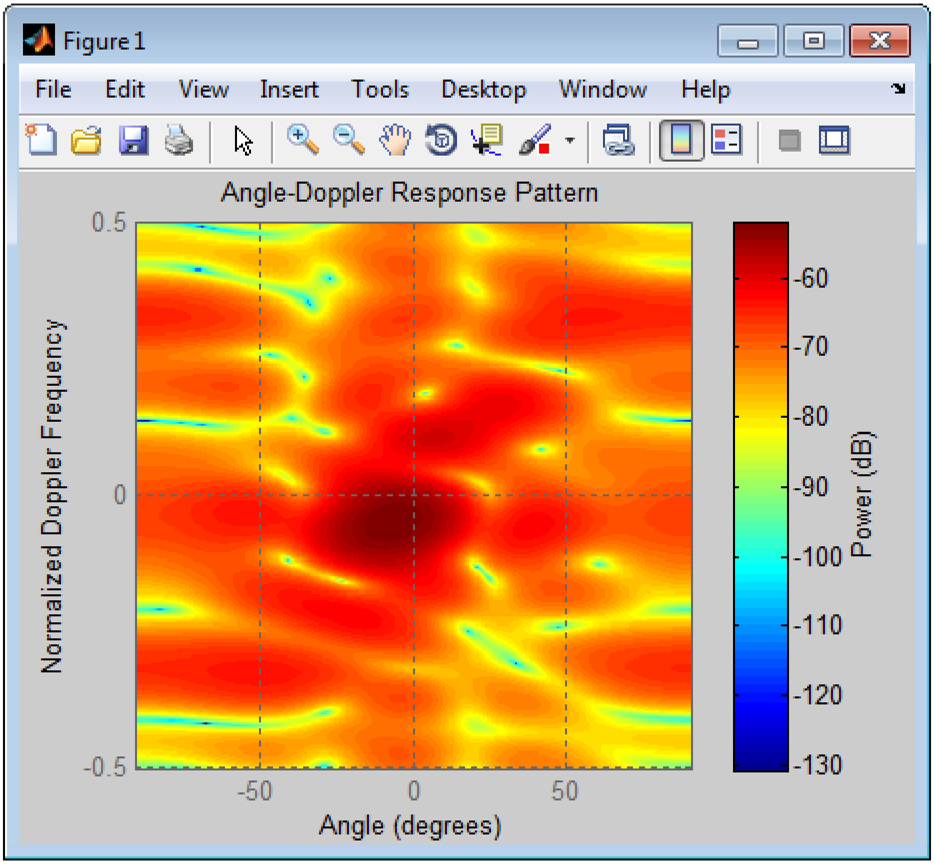


Figure 1
File Edit View Insert Tools Desktop Window Help
Angle-Doppler Response Pattern
Normalized Doppler Frequency
0.5
0
-0.5
-50
0
50
Angle (degrees)
-60
-70
-80
-90
-100
-110
-120
-130
Power (dB)

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);
```

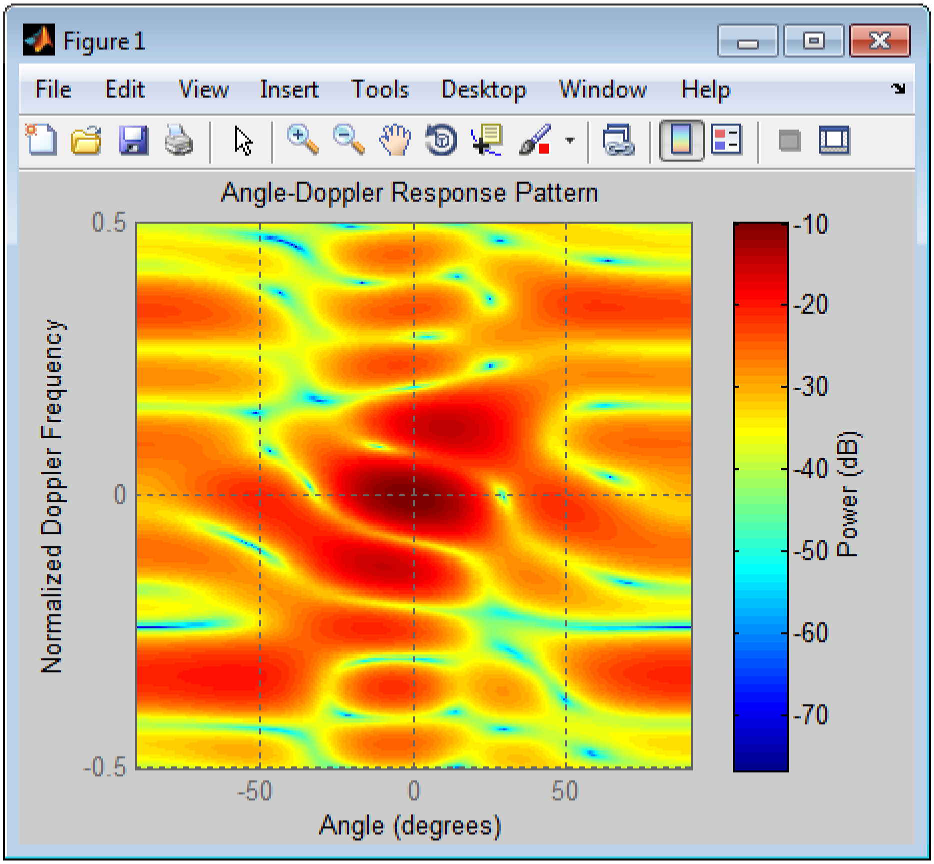


Figure 1
File Edit View Insert Tools Desktop Window Help
Angle-Doppler Response Pattern
Normalized Doppler Frequency
0.5
0
-0.5
-50
0
50
Angle (degrees)
-10
-20
-30
-40
-50
-60
-70
Power (dB)

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 element-wise 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

- Acceleration of Clutter Simulation Using GPU and Code Generation
- 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.

phased.gpu.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.

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);
```

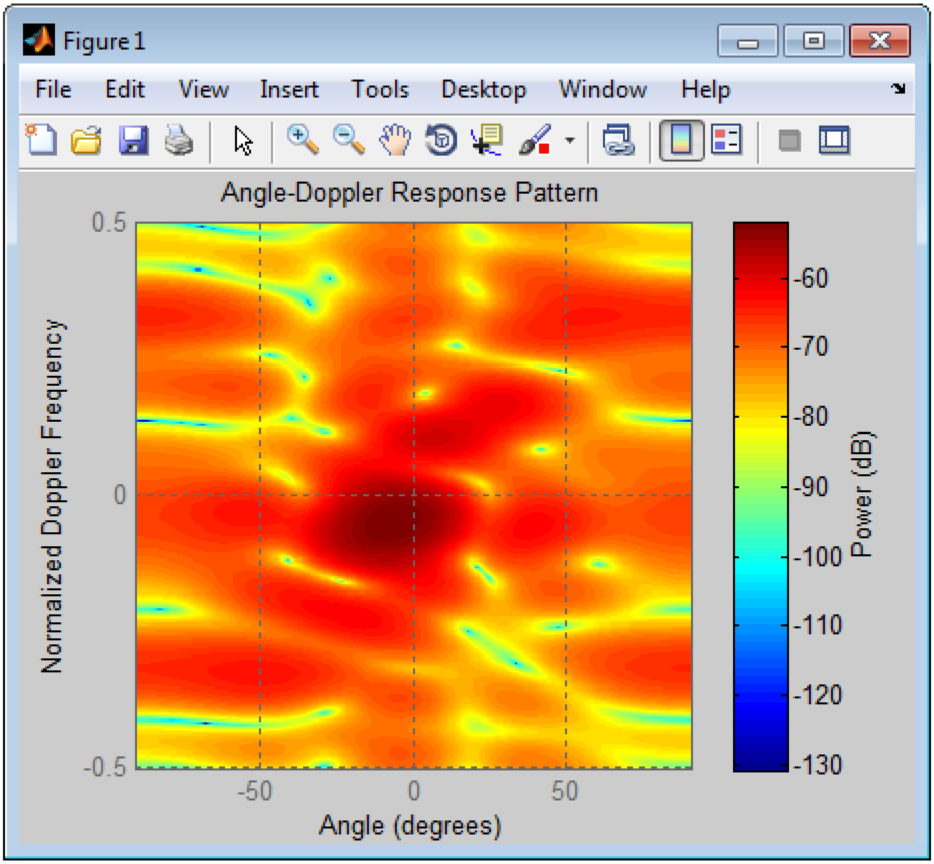


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Angle-Doppler Response Pattern
Normalized Doppler Frequency
0.5
0
-0.5
-50
0
50
Angle (degrees)
-60
-70
-80
-90
-100
-110
-120
-130
Power (dB)

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);
```

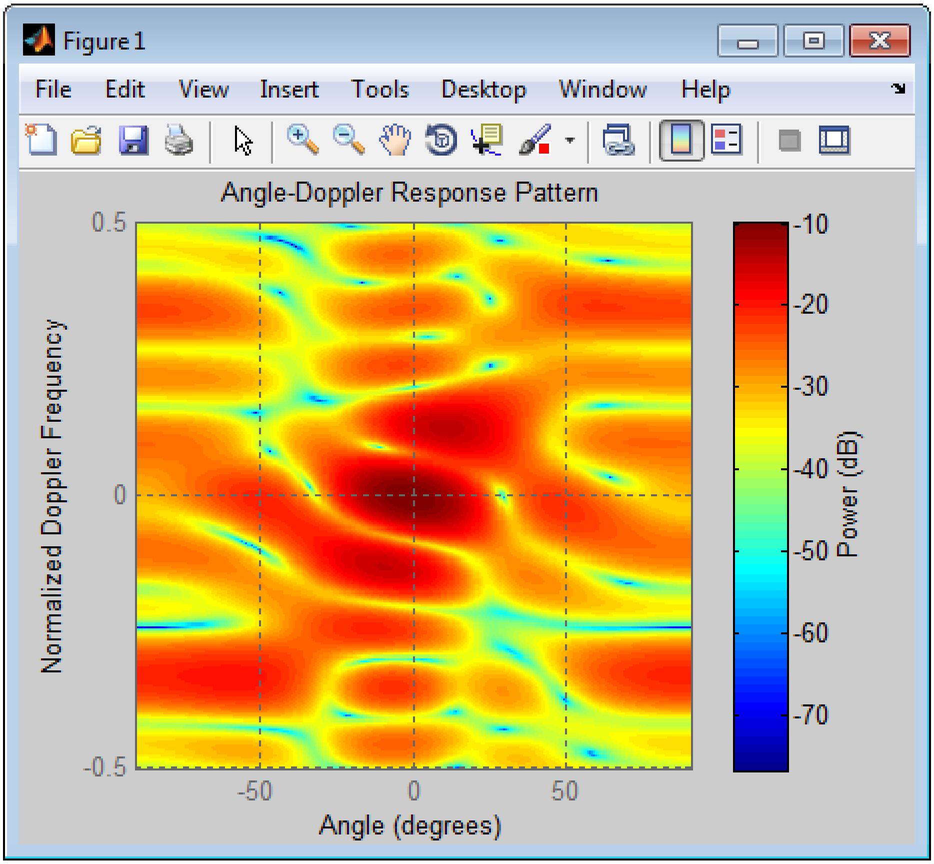


Figure 1
File Edit View Insert Tools Desktop Window Help
Angle-Doppler Response Pattern
0.5
0
-0.5
Normalized Doppler Frequency
-50
0
50
Angle (degrees)
-10
-20
-30
-40
-50
-60
-70
Power (dB)

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

- Acceleration of Clutter Simulation Using GPU and Code Generation
- Ground Clutter Mitigation with Moving Target Indication (MTI) Radar

Concepts

- “Clutter Modeling”
- “GPU Computing”

Purpose

Heterogeneous conformal array

Description

The `HeterogeneousConformalArray` object constructs a conformal array from a heterogeneous set of antenna elements. A heterogeneous array is an array in which the antenna or microphone elements may be of different kinds or have different properties. An example would be an array of elements each having different antenna patterns. A *conformal array* can have elements in any position pointing in any direction.

To compute the response for each element in the array for specified directions:

1. Define and set up your conformal array. See "Construction" on page 1-468.
2. Call `step` to compute the response according to the properties of `phased.HeterogeneousConformalArray`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.HeterogeneousConformalArray` creates a heterogeneous conformal array System object, `H`. This object models a heterogeneous conformal array formed with sensor elements whose pattern may vary from element to element.

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

Properties

ElementSet

Set of elements used in the array

Specify the set of different elements used in the sensor array as a row MATLAB cell array. Each member of the cell array contains an element object in the phased package. Elements specified in the `ElementSet` property must be either all antennas or all microphones. In addition, all specified antenna elements should have same polarization capability. Specify the element of the sensor array as a handle. The element must be an element object in the `phased` package.

Default: One cell containing one isotropic antenna element

ElementIndices

Elements location assignment

This property specifies the mapping of elements in the array. The property assigns elements to their locations in the array using the indices into the `ElementSet` property. The value of `ElementIndices` must be an length-N row vector. In this vector, N represents the number of elements in the array. The values in the vector specified by `ElementIndices` should be less than or equal to the number of entries in the `ElementSet` property.

Default: `[1 2 2 1]`

ElementPosition

Element positions

`ElementPosition` specifies the positions of the elements in the conformal array. The value of the `ElementPosition` property must be a 3-by-N matrix, where N indicates the number of elements in the conformal array. Each column of `ElementPosition` represents the position, in the form `[x; y; z]` (in meters), of a single element in the array's local coordinate system. The local coordinate system has its origin at an arbitrary point.

Default: `[0; 0; 0]`

ElementNormal

Element normal directions

`ElementNormal` specifies the normal directions of the elements in the conformal array. Angle units are degrees. The value assigned to `ElementNormal` must be either a 2-by-N matrix or a 2-by-1 column vector. The variable N indicates the number of elements in the array. If the value of `ElementNormal` is a matrix, each column specifies the normal direction of the corresponding element in the form `[azimuth;elevation]` with respect to the local coordinate system. The local coordinate system aligns the positive x-axis with the direction normal to the conformal array. If the value of `ElementNormal` is a 2-by-1 column vector, it specifies the pointing direction of all elements in the array.

You can use the `ElementPosition` and `ElementNormal` properties to represent any arrangement in which pairs of elements differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

Default: `[0; 0]`

Taper

Element taper or weighting

Element tapering specified as a complex-valued scalar or a complex-valued 1-by-N row vector. N is the number of elements in the array as determined by the size of the `ElementIndices` property. Tapers, also known as weights, are applied to each sensor element in the sensor array and modify both the amplitude and phase of the received data. If `'Taper'` is a scalar, the same weights are applied to each element. If `'Taper'` is a vector, each weight is applied to the corresponding sensor element.

Default: 1

Methods

clone	Create system object with identical 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
getTaper	Array element tapers
isLocked	Locked status for input attributes and nontunable properties
isPolarizationCapable	Polarization capability
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

Heterogeneous Uniform Circular Array (UCA)

Construct an 8-element heterogeneous uniform circular array (UCA). Four of the elements have a cosine pattern with a power of 1.6. The remaining four have a cosine pattern with a power of 1.5. Plot its response as a function of elevation angle. Assume a 1 GHz operating frequency. The wave propagation speed is the speed of light.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.6);
sElement2 = phased.CosineAntennaElement('CosinePower',1.5);
sArray = phased.HeterogeneousConformalArray(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
N = 8; azang = (0:N-1)*360/N-180;
sArray.ElementPosition = ...
    [cosd(azang);sind(azang);zeros(1,N)];
sArray.ElementNormal = [azang;zeros(1,N)];
c = physconst('LightSpeed');
fc = 1e9;
plotResponse(sArray,fc,c,'RespCut','El','Format','Polar');
```

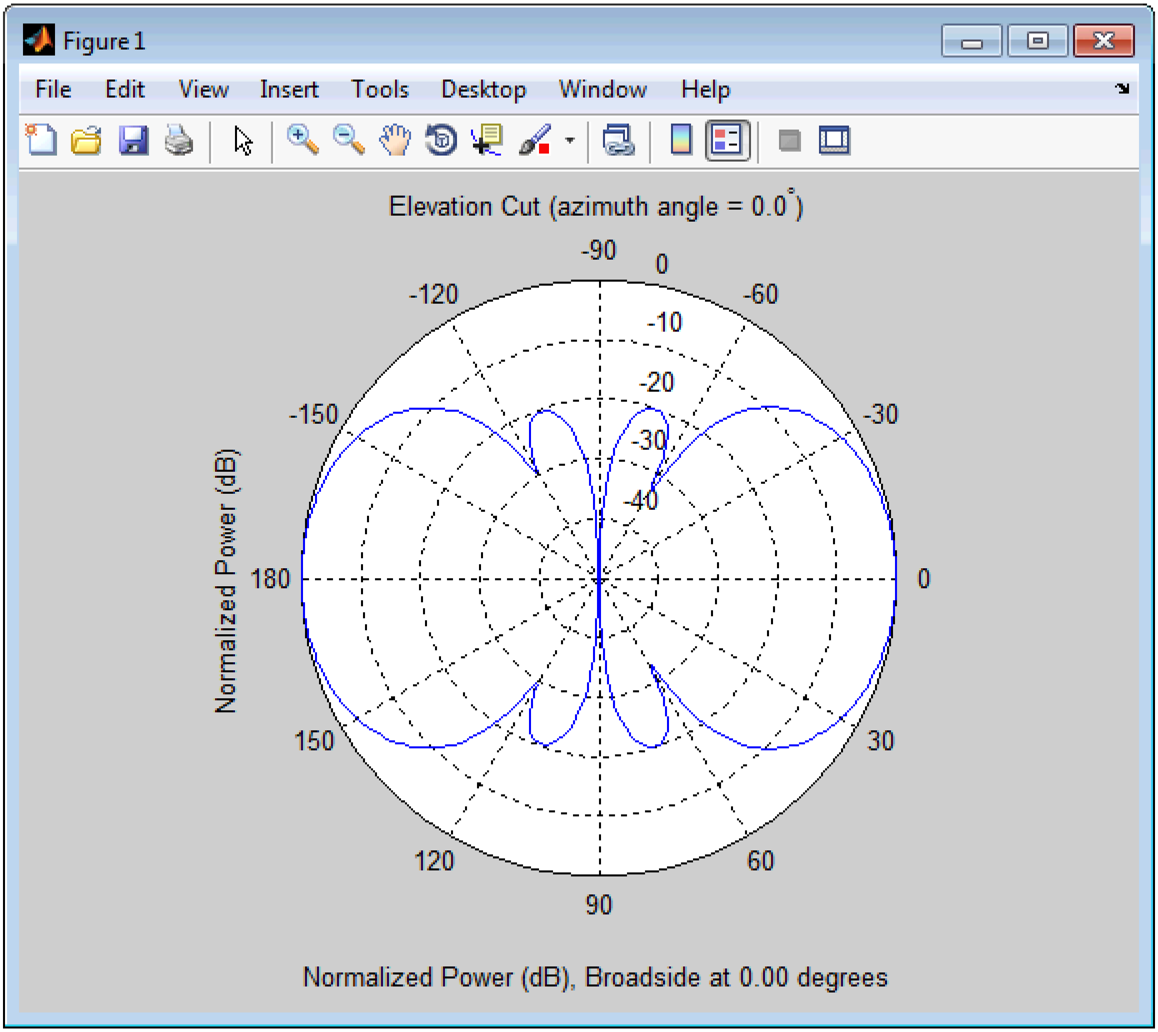


Figure 1
File Edit View Insert Tools Desktop Window Help
Elevation Cut (azimuth angle = 0.0°)
-90
0
-120
-60
-10
-20
-150
-30
-30
-40
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

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` **|** `phased.HeterogeneousULA` **|** `phased.HeterogeneousURA` **|** `phased.ConformalArray` **|** `uv2azel` **|** `phitheta2azel`

Related Examples

- Phased Array Gallery

Purpose Create system object with identical 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 an 8-element uniform circular array

The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8; azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    [cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)],...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
c = physconst('LightSpeed');
y = collectPlaneWave(sArray,randn(4,2),[10 30],c);
disp(y(:,1:2));
```

```
   0.3237 + 0.4890i   0.6039 + 0.0301i
   0.6786 - 0.7586i  -0.5528 + 1.0947i
   1.8804 + 0.6692i   1.2940 + 1.4305i
   2.4967 + 1.3510i   2.1896 + 1.6319i
```

Algorithms

`collectPlaneWave` modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. The method does not account for the response of individual elements in the array.

For further details, see Van Trees [1].

References

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

See Also

`uv2azel` | `phitheta2azel`

phased.HeterogeneousConformalArray.getElementPositio

Purpose

Positions of array elements

Syntax

```
POS = getElementPosition(H)
POS = getElementPosition(H,ELEIDX)
```

Description

`POS = getElementPosition(H)` returns the element positions of the HeterogeneousConformalArray System object, `H`. `POS` is an 3-by-N matrix where N is the number of elements in `H`. Each column of `POS` defines the position of an element in the local coordinate system, in meters, in the form `[x; y; z]`.

For details regarding the local coordinate system of the conformal or heterogeneous conformal array, enter `phased.ConformalArray.coordinateSystemInfo`.

`POS = getElementPosition(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.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
N = 8; azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    [cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)],...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
pos = getElementPosition(sArray);
disp(pos(:,1:4));
```

```
  -1.0000    -0.7071          0     0.7071
   0         -0.7071    -1.0000    -0.7071
```

```
         0             0        0        0
```

phased.HeterogeneousConformalArray.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 heterogeneous 8-element uniform circular array and show that `getNumElements` returns 8.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
N = 8; azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    [cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal',[azang;zeros(1,N)],...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
N = getNumElements(sArray)
```

```
N =

     8
```

phased.HeterogeneousConformalArray.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.HeterogeneousConformalArray.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.HeterogeneousConformalArray.getTaper

Purpose

Array element tapers

Syntax

`wts = getTaper(h)`

Description

`wts = getTaper(h)` returns the tapers applied to each element of a conformal array, `h`. Tapers are often referred to as weights.

Input Arguments

h - Conformal array

`phased.HeterogeneousConformalArray` System object

Conformal array specified as a `phased.HeterogeneousConformalArray` System object.

Output Arguments

wts - Array element tapers

N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1, complex-valued vector, where *N* is the number of elements in the array.

Examples

Construct a 12-element, 2-ring tapered disk array where the outer ring is more heavily tapered than the inner ring.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
elemAngles = ([0:5]*360/6);
elemPosInner = 0.5*[zeros(size(elemAngles));...
    cosd(elemAngles); sind(elemAngles)];
elemPosOuter = [zeros(size(elemAngles));...
    cosd(elemAngles); sind(elemAngles)];
elemNorms = repmat([0;0],1,12);
taper =  [ones(size(elemAngles)),...
    0.3*ones(size(elemAngles))];
sArray = phased.HeterogeneousConformalArray(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 1 1 2 2 2 2 2 2],...
    'ElementPosition',[elemPosInner,elemPosOuter],...
    'ElementNormal',elemNorms,...
    'Taper',taper);
w = getTaper(sArray)
```

List the taper values.

```
w =

    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    1.0000
    0.3000
    0.3000
    0.3000
    0.3000
    0.3000
    0.3000
```

Draw the array showing taper colors.

```
viewArray(sArray,'ShowTaper',true,'ShowIndex','all');
```

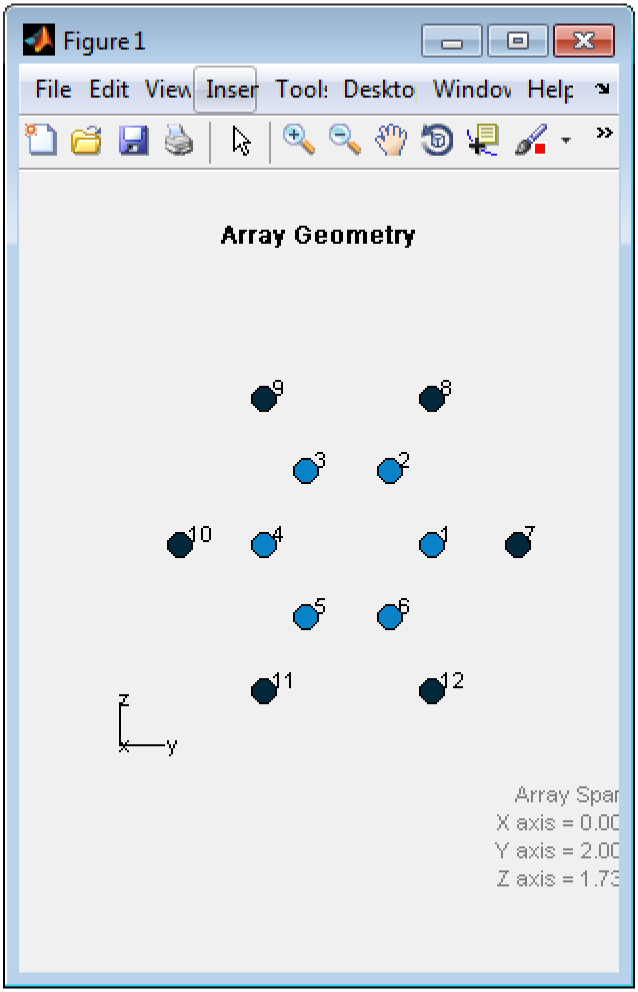


Figure 1
File Edit View Insert Tools Desktop Window Help
Array Geometry
9
8
3
2
10
4
1
7
5
6
11
12
z
x
y
Array Span
X axis = 0.00
Y axis = 2.00
Z axis = 1.73

phased.HeterogeneousConformalArray.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 `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.HeterogeneousConformalArray.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h - Conformal array

Conformal array specified as a `phased.HeterogeneousConformalArray` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

Conformal Array of Short-dipole Antenna Elements Supports Polarization

Show that a circular conformal array of `phased.ShortDipoleAntennaElement` antenna elements supports polarization.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
elemAngles = ([0:5]*360/6);
elemPosInner = 0.5*[zeros(size(elemAngles));...
    cosd(elemAngles); sind(elemAngles)];
elemPosOuter = [zeros(size(elemAngles));...
    cosd(elemAngles); sind(elemAngles)];
elemNorms = repmat([0;0],1,12);
sArray = phased.HeterogeneousConformalArray(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 1 1 2 2 2 2 2 2],...
    'ElementPosition',[elemPosInner,elemPosOuter],...
    'ElementNormal',elemNorms);
isPolarizationCapable(sArray)

ans =

     1
```

The returned value `true (1)` shows that this array supports polarization.

phased.HeterogeneousConformalArray.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 specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the array response pattern. The allowable values are |`'None' | 'Combined' | 'H' | 'V' |` where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'Weights'

Weight values applied to the array, specified as a length-N column vector or N-by-M matrix. The dimension N is the number of elements in the array. The interpretation of M depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-M row vector	Apply one set of weights for the same single frequency or all M frequencies.
N-by-M matrix	Scalar	Apply all of the M different columns in `Weights` for the same single frequency.
	1-by-M row vector	Apply each of the M different columns in `Weights` for the corresponding frequency in `FREQ`.

'AzimuthAngles'

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the U coordinates for visualizing the radiation pattern in U/V space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

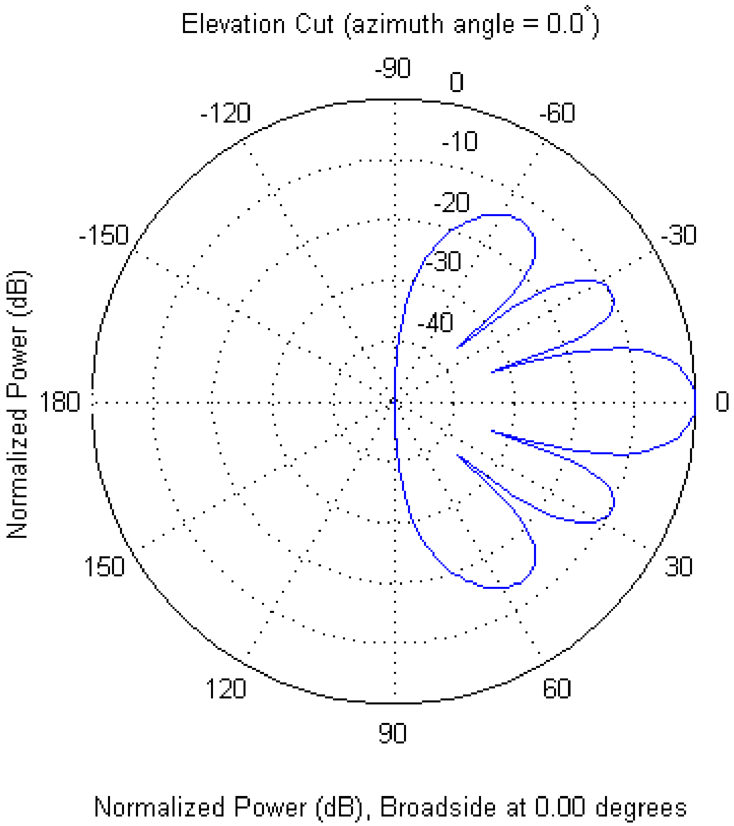
Plot Response and Directivity of 8-Element Uniform Circular Array

This example shows how to construct an 8-element uniform circular array (UCA) with two different antenna patterns.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8; azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    0.4*[zeros(1,N);cosd(azang);sind(azang)],...
    'ElementNormal', zeros(2,N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
```

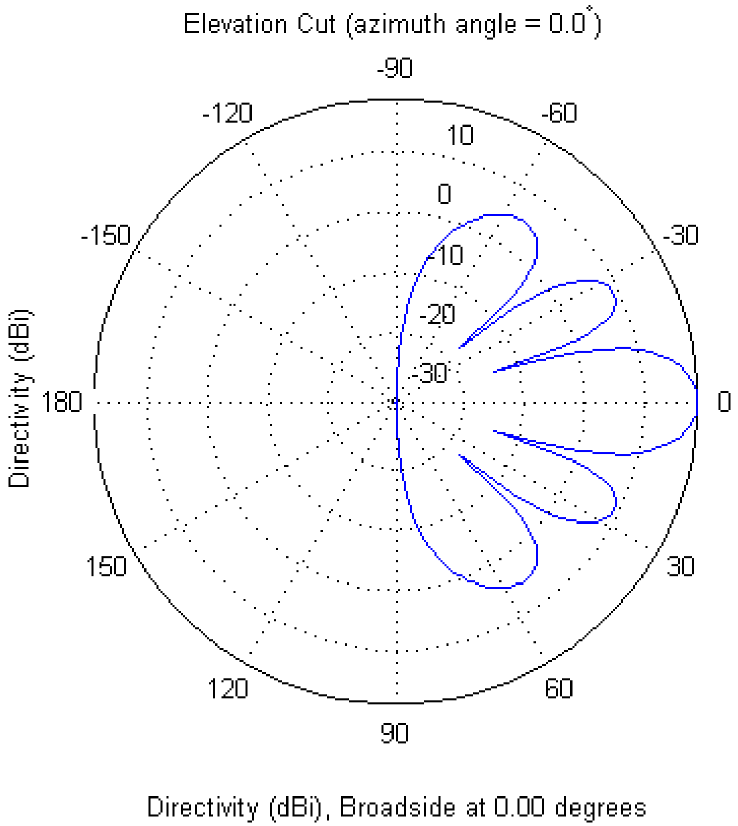
```
Plot its elevation response. Assume the operating frequency
is 1 GHz and the wave propagation speed is the speed of light.
```

```
c = physconst('LightSpeed');
fc = 1e9;
plotResponse(sArray,fc,c,'RespCut','El','Format','Polar');
```



Plot the directivity.

```
plotResponse(sArray,fc,c,'RespCut','El','Format','Polar',...
    'Unit','dbi');
```



Plot Response of Disk Array

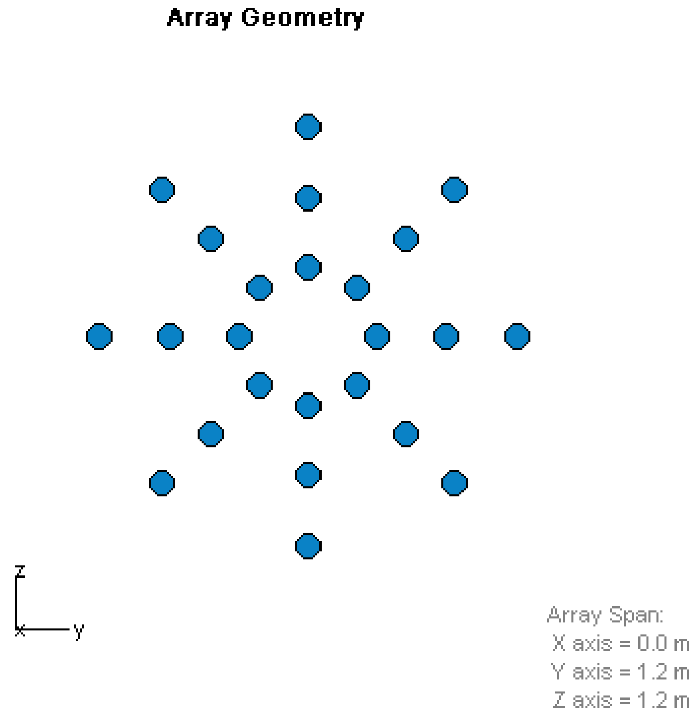
This example shows how to construct a 24-element disk array using elements with two different antenna patterns and plot its response.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8; azang = (0:N-1)*360/N-180;
p0 = [zeros(1,N);cosd(azang);sind(azang)];
posn = [0.6*p0, 0.4*p0, 0.2*p0];
```

```
sArray1 = phased.HeterogeneousConformalArray(...
    'ElementPosition',posn,...
    'ElementNormal', zeros(2,3*N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 1 1 1 1,...
    1 1 1 1 1 1 1 1,...
    2 2 2 2 2 2 2 2]);
```

Show the array.

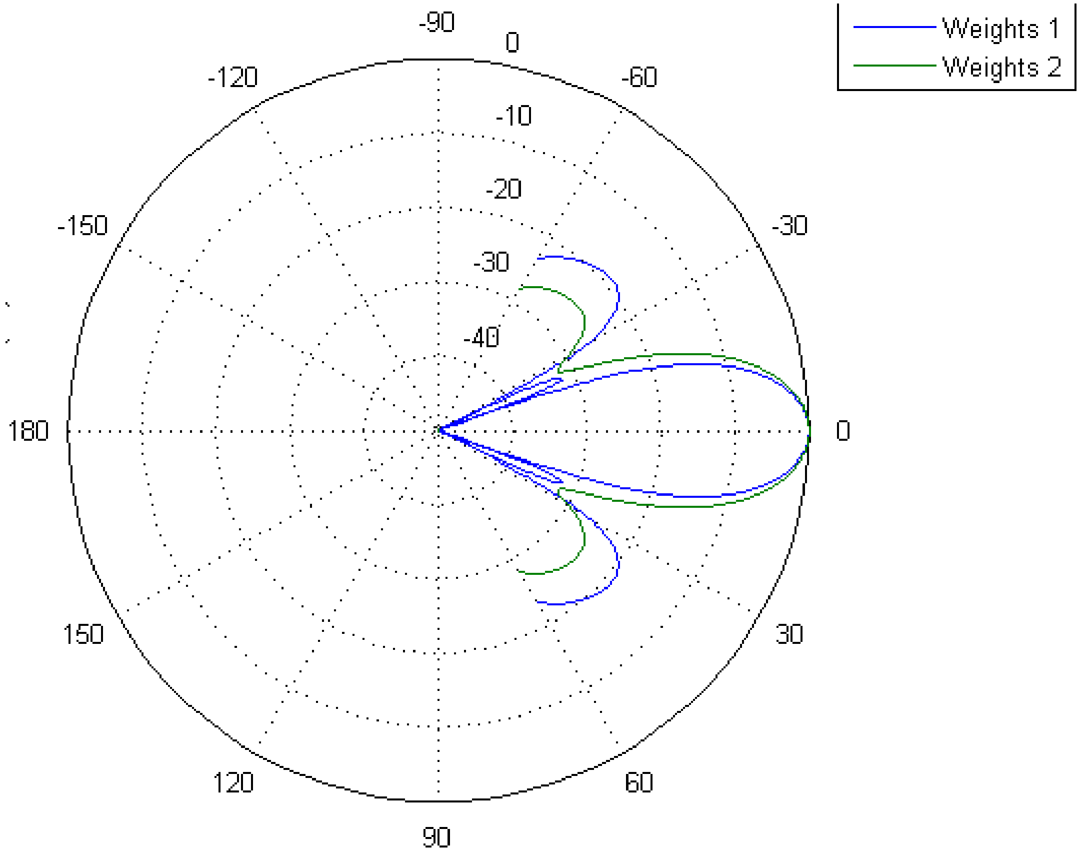
```
viewArray(sArray1);
```



Plot the elevation response of this array using uniform weights on the elements and also a tapered set of weights set by the `Weights` parameter. Using the `ElevationAngles` parameter, restrict the plot of the response from -60 to 60 degrees in 0.1 degree increments. Assume the operating frequency is 1 GHz and the wave propagation speed is the speed of light.

```
c = physconst('LightSpeed');
fc = 1e9;
wts1 = ones(3*N,1);
```

```
wts1 = wts1/sum(abs(wts1));
wts2 = [0.5*ones(N,1); 0.7*ones(N,1); 1*ones(N,1)];
wts2 = wts2/sum(abs(wts2));
plotResponse(sArray1,fc,c,'RespCut','El',...
     'Format','Polar',...
     'ElevationAngles',[-60:0.1:60],...
     'Weights',...
     [wts1,wts2],...
     'Unit','db');
```



As expected, the tapered weights broaden the mainlobe and reduce the sidelobes.

See Also

`uv2azel` | `azel2uv`

phased.HeterogeneousConformalArray.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`

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions N-by-M-by-L. N is the number of elements in the array. The dimension M is the number of angles specified in `ANG`. L is the number of frequencies specified in `FREQ`. For any element, the columns of `RESP` contain the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.
- If the array is capable of supporting polarization, the voltage response, `RESP`, is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions N-by-M-by-L. N is the number of elements in the array, and M is the number of angles specified in `ANG`. L is the number of frequencies specified in `FREQ`. Each column of `RESP` contains the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.

Examples

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° azimuth and 5°.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8; azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    [cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal', zeros(2,N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
fc = 1e9;
ang = [30;5];
resp = step(sArray,fc,ang)

resp =

    0.8013
    0.8013
    0.8013
    0.8013
    0.7666
    0.7666
    0.7666
    0.7666
```

See Also

`uv2azel` | `phitheta2azel`

phased.HeterogeneousConformalArray.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. `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`

'ShowTaper'

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

'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 heterogeneous uniform circular array.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
N = 8; azang = (0:N-1)*360/N-180;
sArray = phased.HeterogeneousConformalArray(...
    'ElementPosition',...
    [cosd(azang);sind(azang);zeros(1,N)],...
    'ElementNormal', zeros(2,N),...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1 2 2 2 2]);
```

```
viewArray(sArray,'ShowIndex','all','ShowNormal',true);
```

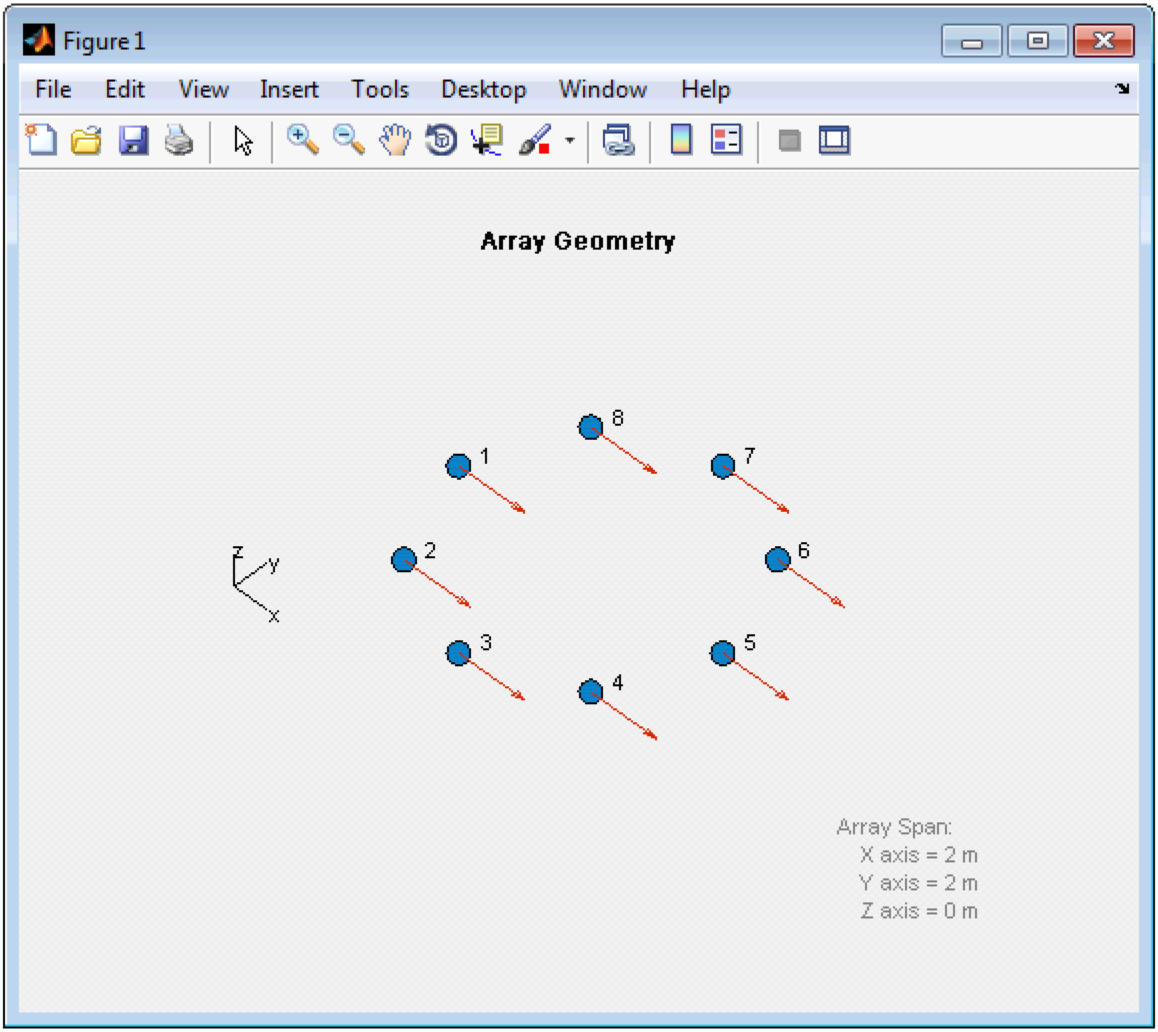


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Array Geometry
8
1
7
2
6
3
5
4
z
y
x
Array Span:
X axis = 2 m
Y axis = 2 m
Z axis = 0 m

See Also `phased.ArrayResponse`

Related Examples

- Phased Array Gallery

Purpose

Heterogeneous uniform linear array

Description

The `phased.HeterogeneousULA` object creates a uniform linear array from a heterogeneous set of antenna elements. A heterogeneous array is an array in which the antenna or microphone elements may be of different kinds or have different properties. An example would be an array of elements each having different antenna patterns.

To compute the response for each element in the array for specified directions:

1. Define and set up your uniform linear array. See "Construction" on page 1-510.
2. Call `step` to compute the response according to the properties of `phased.HeterogeneousULA`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.HeterogeneousULA` creates a heterogeneous uniform linear array (ULA) System object, `H`. The object models a heterogeneous ULA formed with generally different sensor elements. The origin of the local coordinate system is the phase center of the array. The positive *x*-axis is the direction normal to the array, and the elements of the array are located along the *y*-axis.

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

Properties

ElementSet

Set of elements used in the array

Specify the set of different elements used in the sensor array as a row MATLAB cell array. Each member of the cell array contains an element object in the phased package. Elements specified in the `ElementSet` property must be either all antennas or all microphones. In addition, all specified antenna elements should have same polarization capability. Specify the element of the sensor array as a handle. The element must be an element object in the `phased` package.

Default: One cell containing one isotropic antenna element

ElementIndices

Elements location assignment

This property specifies the mapping of elements in the array. The property assigns elements to their locations in the array using indices into the `ElementSet` property. `ElementIndices` must be a 1-by-N row vector where N is greater than 1. N is the number of elements in the sensor array. The values in `ElementIndices` should be less than or equal to the number of entries in the `ElementSet` property.

Default: `[1 1]`

ElementSpacing

Element spacing

A scalar containing the spacing (in meters) between two adjacent elements in the array.

Default: `0.5`

Taper

Element tapering

Element tapering specified as a complex-valued scalar or a complex-valued 1-by-N row vector. N is the number of elements in the array as determined by the size of the `ElementIndices` property. Tapers, also known as weights, are applied to each sensor element in the sensor array and modify both the amplitude and phase of the received data. If `'Taper'` is a scalar, the same weights are applied to each element. If `'Taper'` is a vector, each weight is applied to the corresponding sensor element.

Default: `1`

Methods

clone	Create new system object with identical 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
getTaper	Array element tapers
isLocked	Locked status for input attributes and nontunable properties
isPolarizationCapable	Polarization capability
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 10-Element Heterogeneous ULA Array

Create a 10-element heterogeneous ULA consisting of cosine antenna elements with different power factors. Two elements at each end have power values of 1.5 while the inside elements have power values of 1.8. Find the response of each element at boresight.

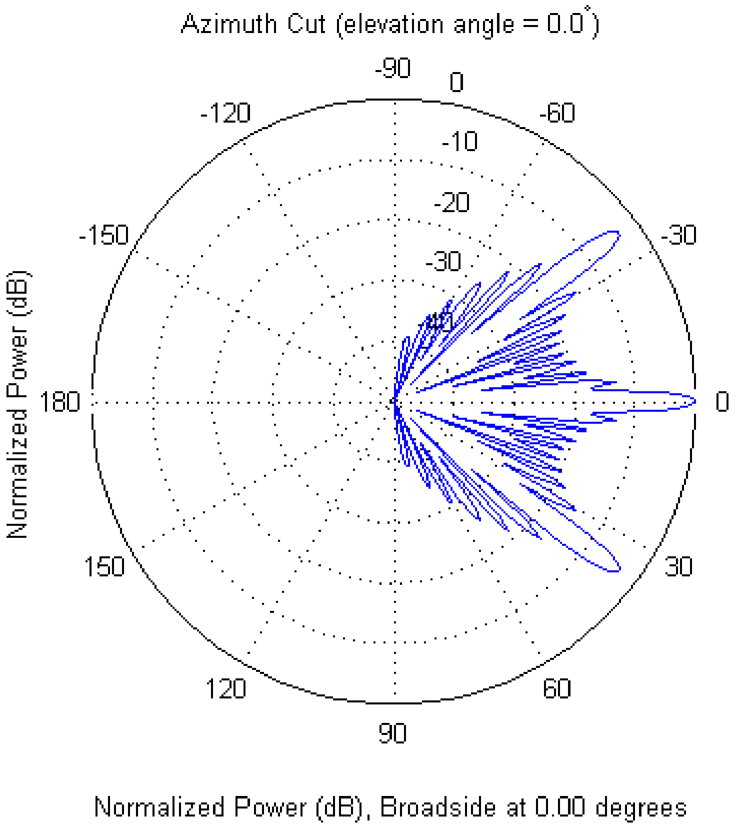
Construct the heterogeneous array and show the element responses at 1 GHz.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 2 2 2 2 2 2 1 1 ]);
fc = 1e9;
ang = [0;0];
resp = step(sArray,fc,ang)
```

```
resp =

     1
     1
     1
     1
     1
     1
     1
     1
     1
     1
```

Plot an azimuth cut of the array response at 1 GHz.

```
c = physconst('LightSpeed');
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar');
```



Response of an Array of Polarized Short-Dipole Antennas

Build a heterogeneous uniform line array of 10 short-dipole sensor elements. Because short dipoles support polarization, the array should also. Verify that the array supports polarization by looking at the output of `isPolarizationCapable`. Then, draw the array, showing the tapering.

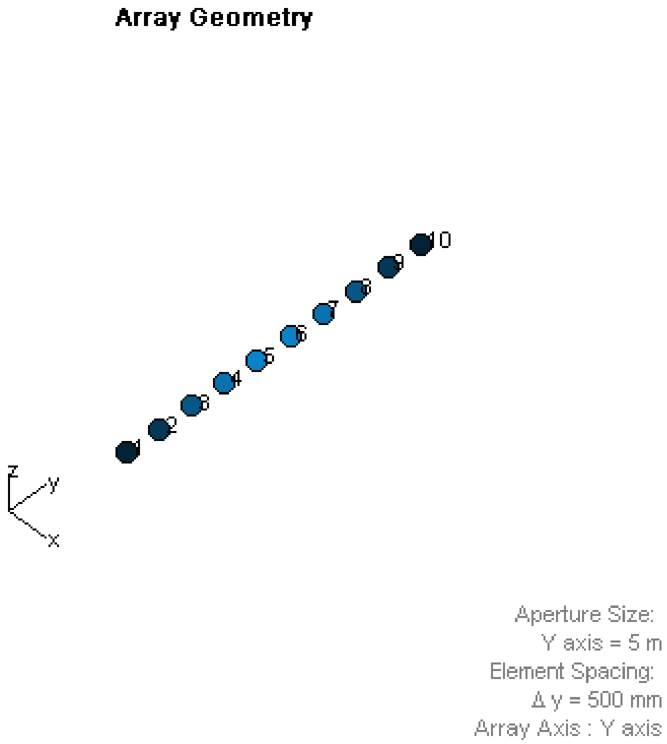
Build the array. Then, verify that it supports polarization by looking at the returned value of the `isPolarizationCapable` method.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 2 2 2 2 2 2 1 1 ],...
    'Taper',taylorwin(10)');
isPolarizationCapable(sArray)
```

```
ans =

     1
```

View the array.

```
viewArray(sArray,'ShowTaper',true,'ShowIndex',...
    'All','ShowTaper',true)
```

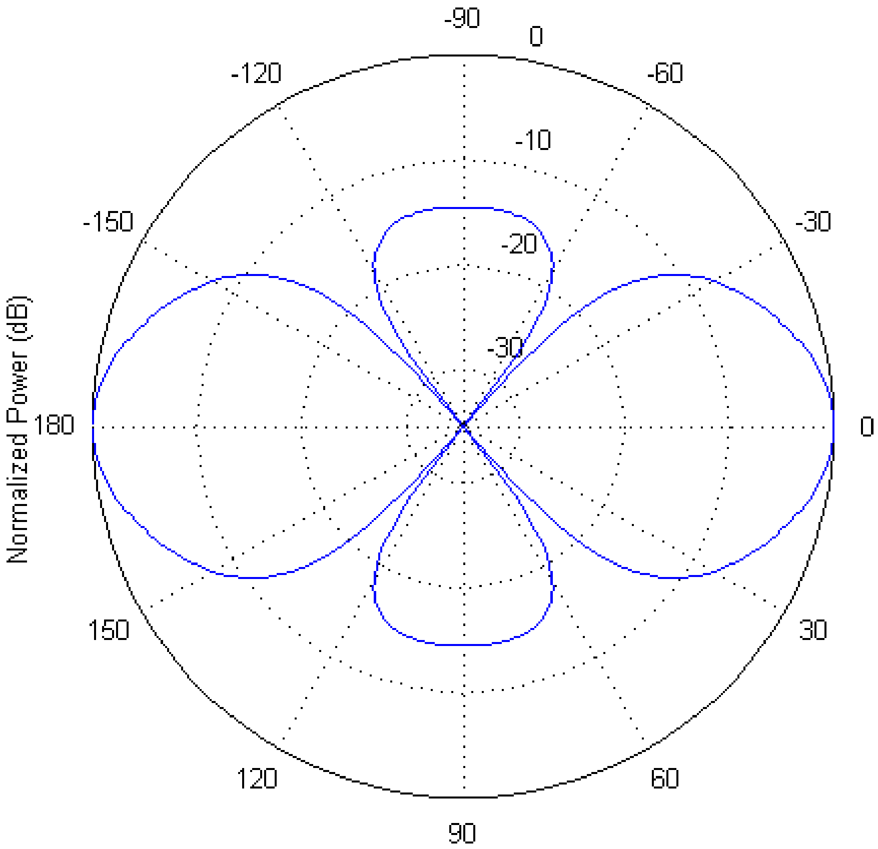


Show the element horizontal polarization responses at 10 degrees azimuth angle.

```
fc = 150e6;
ang = [10];
resp = step(sArray,fc,ang)
resp.H


resp =
    H: [10x1 double]
    V: [10x1 double]


ans =

         0
         0
   -1.2442
   -1.6279
   -1.8498
   -1.8498
   -1.6279
   -1.2442
         0
         0
```

Plot the combined polarization response.

```
c = physconst('LightSpeed');
plotResponse(sArray,fc,c,'RespCut','Az','Format',...
    'Polar','Polarization','C');
```



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.ULA` | `phased.URA` | `phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.HeterogeneousURA` | `phased.CosineAntennaElementphased.CrossedDipoleAntennaElement`

| `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement``phased.ShortDipoleAntennaElement`

Related Examples

- Phased Array Gallery

phased.HeterogeneousULA.clone

Purpose Create new system object with identical 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 heterogeneous 4-element ULA.

The signals arrive from 10° and 30° degrees azimuth. Both signals have an elevation angle of 0°. Assume the propagation speed is the speed of light and the carrier frequency of the signal is 100 MHz.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 1]);
y = collectPlaneWave(sArray,randn(4,2),[10 30],1e8,...
    physconst('LightSpeed'));
y(:,1)
```

```
ans =
   0.7430 - 0.3705i
   0.8418 + 0.4308i
  -2.4817 + 0.9157i
   1.0724 - 0.4748i
```

Algorithms

`collectPlaneWave` modulates the input signal with a phase corresponding to the delay caused by the direction of arrival. The method does not account for the response of individual elements in the array.

For further details, see [1].

References

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

See Also

`uv2azel` | `phitheta2azel`

phased.HeterogeneousULA.getElementPosition

Purpose

Positions of array elements

Syntax

```
POS = getElementPosition(H)
POS = getElementPosition(H,ELEIDX)
```

Description

`POS = getElementPosition(H)` returns the element positions of the HeterogeneousULA System object, `H`. `POS` is a 3-by-N matrix, where N is the number of elements in `H`. Each column of `POS` defines the position of an element in the local coordinate system, in meters, using the form `[x; y; z]`. 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.

`POS = getElementPosition(H,ELEIDX)` returns only the positions of the elements that are specified in the element index vector `ELEIDX`. This syntax can use any of the input arguments in the previous syntax.

Examples

Construct a 4–element heterogeneous ULA, and obtain the element positions.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 1]);
pos = getElementPosition(sArray);
```

```
pos =

         0         0         0         0
   -0.7500   -0.2500    0.2500    0.7500
         0         0         0         0
```

phased.HeterogeneousULA.getNumElements

Purpose Number of elements in array

Syntax

```
N = getNumElements(H)
```

Description `N = getNumElements(H)` returns the number of elements, *N*, in the HeterogeneousULA object `H`.

Examples Construct a default ULA, and obtain the number of elements in that array.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 1]);
N = getNumElements(sArray)

N =

     4
```

phased.HeterogeneousULA.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.HeterogeneousULA.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.HeterogeneousULA.getTaper

Purpose

Array element tapers

Syntax

`wts = getTaper(h)`

Description

`wts = getTaper(h)` returns the tapers, `wts`, applied to each element of the phased heterogeneous uniform line array (ULA), `h`. Tapers are often referred to as weights.

Input Arguments

h - Heterogeneous Uniform line array

`phased.HeterogeneousULA` System object

Heterogeneous uniform line array specified as a `phased.HeterogeneousULA` System object.

Output Arguments

wts - Array element tapers

N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1 complex-valued vector, where *N* is the number of elements in the array.

Examples

Heterogeneous ULA with Taylor Window Taper

Construct a 5-element heterogeneous ULA with a Taylor window taper. Then, obtain the element taper values.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1],'Taper',taylorwin(5)');
w = getTaper(sArray)

w =
    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

phased.HeterogeneousULA.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 `HeterogeneousULA` 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.HeterogeneousULA.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h - Uniform line array

Uniform line array specified as a `phased.HeterogeneousULA` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability flag returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

Heterogeneous ULA of Short-Dipole Antenna Elements Supports Polarization

Show that a heterogeneous array of `phased.ShortDipoleAntennaElement` antenna elements supports polarization.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
isPolarizationCapable(sArray)
```

```
ans =
     1
```

The returned value `true (1)` shows that this array supports polarization.

phased.HeterogeneousULA.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 specified as a scalar or 1-by-K row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the array response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'Weights'

Weight values applied to the array, specified as a length-N column vector or N-by-M matrix. The dimension N is the number of elements in the array. The interpretation of M depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-M row vector	Apply one set of weights for the same single frequency or all M frequencies.
N-by-M matrix	Scalar	Apply all of the M different columns in `Weights` for the same single frequency.
	1-by-M row vector	Apply each of the M different columns in `Weights` for the corresponding frequency in `FREQ`.

'AzimuthAngles'

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Line Plot Showing Multiple Frequencies

Using a line plot, show the azimuth cut response of a 5-element heterogeneous uniform linear array along 0 degrees elevation. The plot shows the responses at operating frequencies of 200 MHz and 400 MHz.

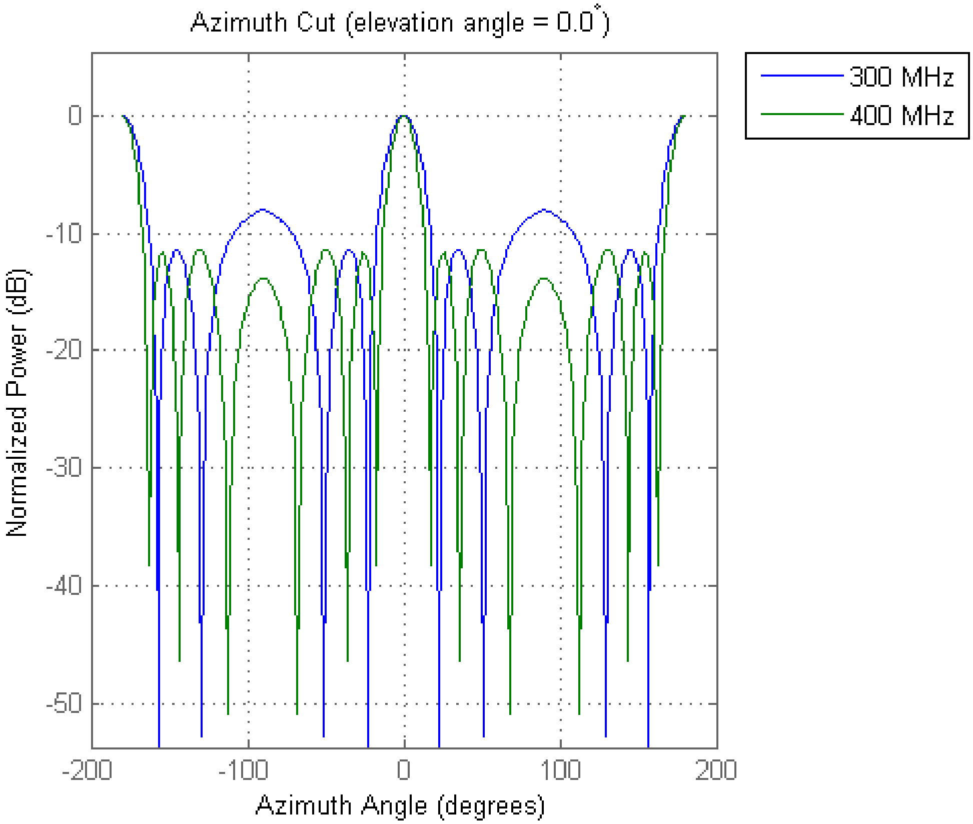
Construct the array from z-directed and y-directed short dipole antenna elements.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
```

Plot the response.

```
fc = [3e8 4e8];
c = physconst('LightSpeed');
```

```
plotResponse(sArray,fc,c);
```



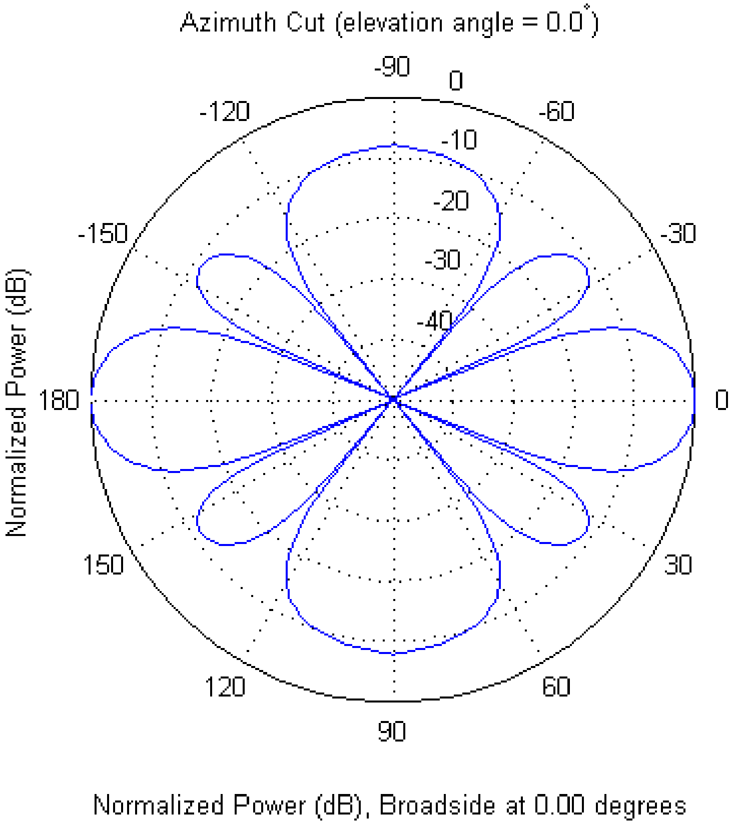
Plot Response and Directivity for 5-Element Array

Construct a 5-element heterogeneous ULA and plot its azimuth response in polar format. Assume the operating frequency spans 200-500 MHz and the wave propagation speed is the speed of light.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
```

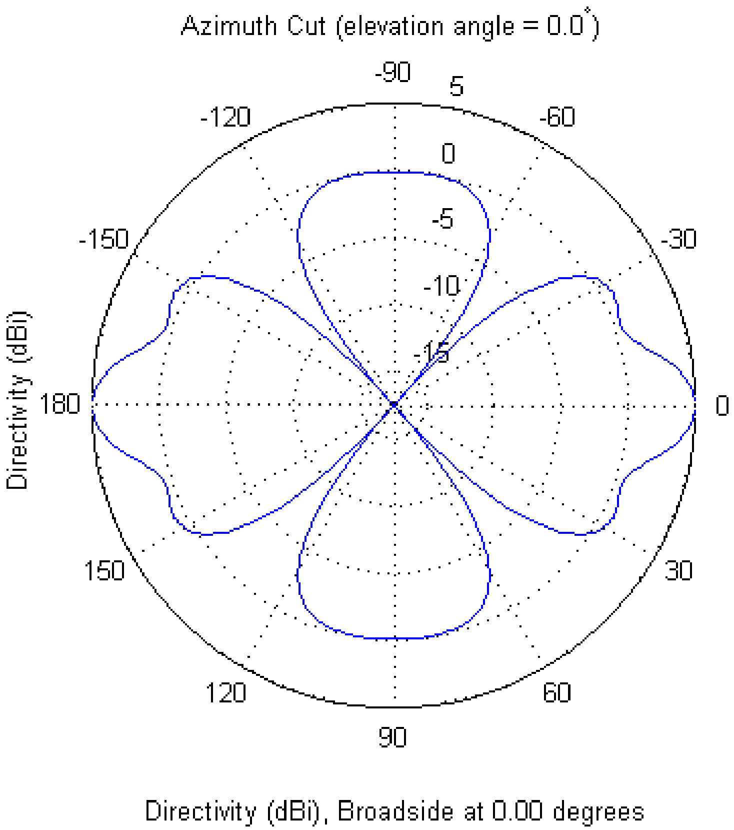
Plot the response at 300 MHz.

```
fc = 3e8;
c = physconst('LightSpeed');
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar');
```



Plot the directivity of the array at 300 MHz.

```
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar',...
    'Unit','dbi');
```



Plot Response for 9-Element Array with Two Weight Sets

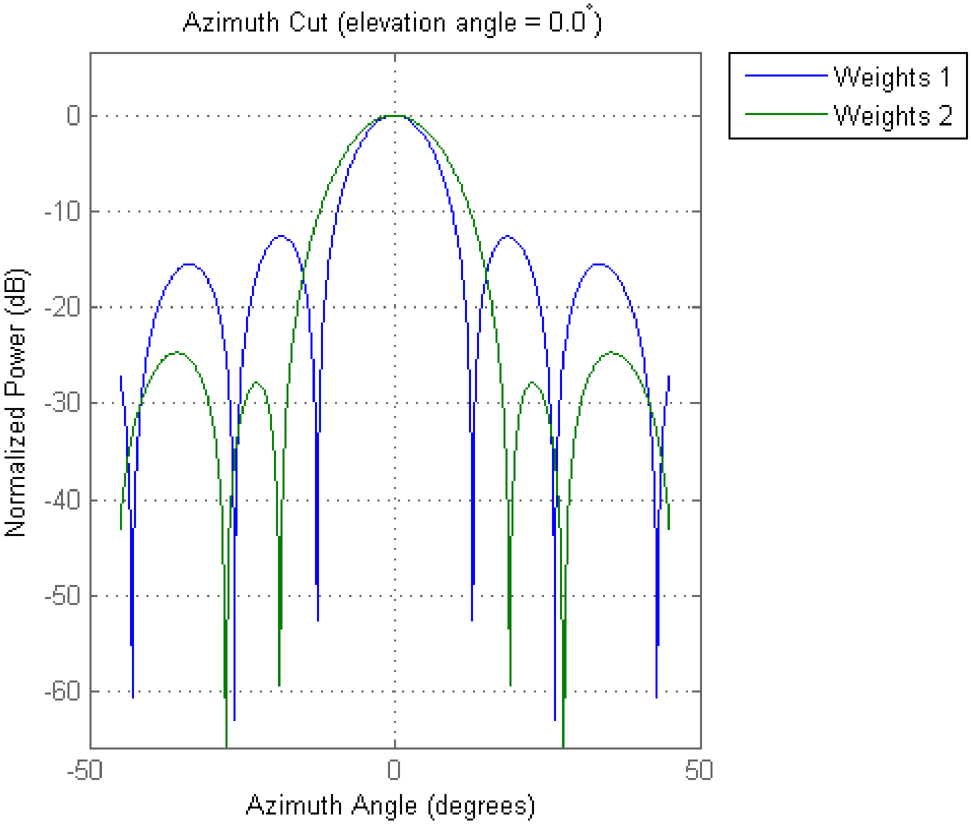
Construct a 9-element heterogeneous ULA from short dipole elements with different orientations. Assume the element response is in the frequency range 200-500 MHz. Use the `Weights` parameter to set two different sets of weights: a uniform weighting and a Taylor weighting. Plot the array's azimuth response in polar format. Using the `AzimuthAngles` parameter, restrict the display range from -45 to 45 degrees in 0.1 degree increments.

Construct the array.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 2 2 2 2 2 1 1]);
```

Plot the response at 300 MHz.

```
fc = 3e8;
wts1 = ones(9,1);
wts2 = taylorwin(9);
c = physconst('LightSpeed');
plotResponse(sArray,fc,c,'RespCut','Az',...
    'AzimuthAngles',[-45:0.1:45],...
    'Weights',[wts1,wts2]);
```



As expected, the tapered weighting broadens the mainlobe and reduces the sidelobes.

See Also

`uv2azel` | `azel2uv`

phased.HeterogeneousULA.release

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.HeterogeneousULA.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

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array. The dimension *M* is the number of angles specified in `ANG`. *L* is the number of frequencies specified in `FREQ`. For any element, the columns of `RESP` contain the responses of the array elements for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.
- If the array is capable of supporting polarization, the voltage response, `RESP`, is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array, and *M* is the number of angles specified in `ANG`. *L* is the number of frequencies specified in `FREQ`. Each column of `RESP` contains the responses of the array elements for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.

Examples

Heterogeneous ULA of Cosine Antenna Elements

Create a 5-element heterogeneous ULA of cosine antenna elements with difference responses, and find the response of each element at 30° azimuth.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
fc = 1e9;
c = physconst('LightSpeed');
ang = [30;0];
resp = step(sArray,fc,ang)
```

```
resp =

    0.8059
    0.7719
    0.7719
    0.7719
    0.8059
```

Response of Heterogeneous Microphone ULA Array

Find the response of a heterogeneous ULA array of 7 custom microphone elements with different responses.

```
sMic1 = phased.CustomMicrophoneElement(...
    'FrequencyResponse',[20 20e3]);
sMic1.PolarPatternFrequencies = [500 1000];
sMic1.PolarPattern = mag2db([...
    0.5+0.5*cosd(sMic1.PolarPatternAngles);...
    0.6+0.4*cosd(sMic1.PolarPatternAngles)]);
sMic2 = phased.CustomMicrophoneElement(...
    'FrequencyResponse',[20 20e3]);
sMic2.PolarPatternFrequencies = [500 1000];
```

```
sMic2.PolarPattern = mag2db([...
    ones(size(sMic2.PolarPatternAngles));...
    ones(size(sMic2.PolarPatternAngles))]);
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sMic1,sMic2},...
    'ElementIndices',[1 1 2 2 2 1 1]);
fc = [1500, 2000];
ang = [40 50; 0 0];
resp = step(sArray,fc,ang)

resp(:,:,1) =

    9.0642    8.5712
    9.0642    8.5712
   10.0000   10.0000
   10.0000   10.0000
   10.0000   10.0000
    9.0642    8.5712
    9.0642    8.5712

resp(:,:,2) =

    9.0642    8.5712
    9.0642    8.5712
   10.0000   10.0000
   10.0000   10.0000
   10.0000   10.0000
    9.0642    8.5712
    9.0642    8.5712
```

See Also

`uv2azel` | `phitheta2azel`

phased.HeterogeneousULA.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. `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`

'ShowTaper'

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

'Title'

String specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

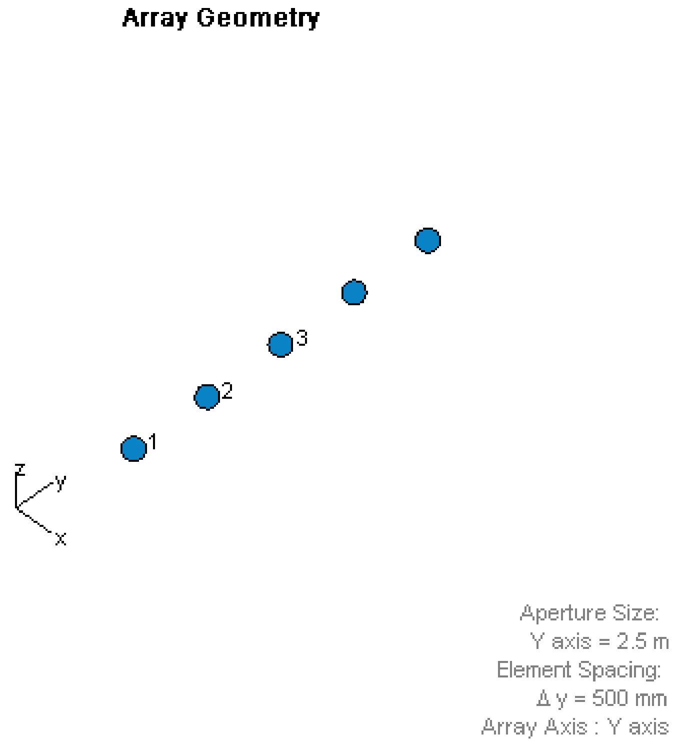
Handle of array elements in figure window.

Examples

Geometry and Indices of Heterogeneous ULA Elements

Display the geometry of a 5-element heterogeneous ULA, and show the indices for the first three elements.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousULA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1]);
viewArray(sArray,'ShowIndex',[1:3]);
```


Array Geometry
1
2
3
z
y
x
Aperture Size:
Y axis = 2.5 m
Element Spacing:
Δ y = 500 mm
Array Axis : Y axis

See Also `phased.ArrayResponse`

Related Examples

- Phased Array Gallery

Purpose

Heterogeneous uniform rectangular array

Description

The `HeterogeneousURA` object constructs a heterogeneous uniform rectangular array (URA).

To compute the response for each element in the array for specified directions:

1. Define and set up your uniform rectangular array. See "Construction" on page 1-553.
2. Call `step` to compute the response according to the properties of `phased.HeterogeneousURA`. The behavior of `step` is specific to each object in the toolbox.

Construction

`H = phased.HeterogeneousURA` creates a heterogeneous uniform rectangular array (URA) System object, `H`. This object models a heterogeneous URA formed with sensor elements whose pattern may vary from element to element. Array elements are distributed in the *yz*-plane in a rectangular lattice. An *M*-by-*N* heterogeneous URA has *M* rows and *N* columns. The array boresight direction is along the positive *x*-axis. The default array is a 2-by-2 URA of isotropic antenna elements.

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

Properties

ElementSet

Set of elements used in the array

Specify the set of different elements used in the sensor array as a row MATLAB cell array. Each member of the cell array contains an element object in the phased package. Elements specified in the `ElementSet` property must be either all antennas or all microphones. In addition, all specified antenna elements should have same polarization capability. Specify the element of the sensor array as a handle. The element must be an element object in the `phased` package.

Default: One cell containing one isotropic antenna element

ElementIndices

Elements location assignment

This property specifies the mapping of elements in the array. The property assigns elements to their locations in the array using the indices into the `ElementSet` property. The value of `ElementIndices` must be an M-by-N matrix. In this matrix, M represents the number of rows and N represents the number of columns. Rows are along y-axis and columns are along z-axis of the local coordinate system. The values in the matrix specified by `ElementIndices` should be less than or equal to the number of entries in the `ElementSet` property.

Default: `[1 1;1 1]`

ElementSpacing

Element spacing

A 1-by-2 vector or a scalar containing the element spacing (in meters) of the array. If `ElementSpacing` is a 1-by-2 vector, it is in the form of `[SpacingBetweenRows,SpacingBetweenColumns]`. See "Spacing Between Columns" on page 1-556 and "Spacing Between Rows" on page 1-556. If `ElementSpacing` is a scalar, both spacings are the same.

Default: `[0.5 0.5]`

Lattice

Element lattice

Specify the element lattice as one of `'Rectangular' | 'Triangular'`. When you set the Lattice property to

`'Rectangular'`, all elements in the heterogeneous URA are aligned in both row and column directions. When you set the `Lattice` property to `'Triangular'`, the elements in even rows are shifted toward the positive row axis direction by a distance of half the element spacing along the row.

Default: `'Rectangular'`

Taper

Element taper

Element tapering specified as a complex-valued scalar or a complex-valued M-by-N matrix. M is the number of elements along the z-axis and N is the number of elements along y-axis. M and N correspond to the values of `[NumberofRows, NumberOfColumns]` in the `Size` property. Tapers, also known as weights, are applied to each sensor element in the sensor array and modify both the amplitude and phase of the received data. If `'Taper'` is a scalar, the same weights are applied to each element. If `'Taper'` is a vector, each weight is applied to the corresponding sensor element.

Default: 1

Methods

clone	Create new system object with identical 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

getTaper	Array element tapers
isLocked	Locked status for input attributes and nontunable properties
isPolarizationCapable	Polarization capability
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 Between Columns

The spacing between columns is the distance between adjacent elements in the same row.

Spacing Between Rows

The spacing between rows is the distance along the column axis direction between adjacent rows.

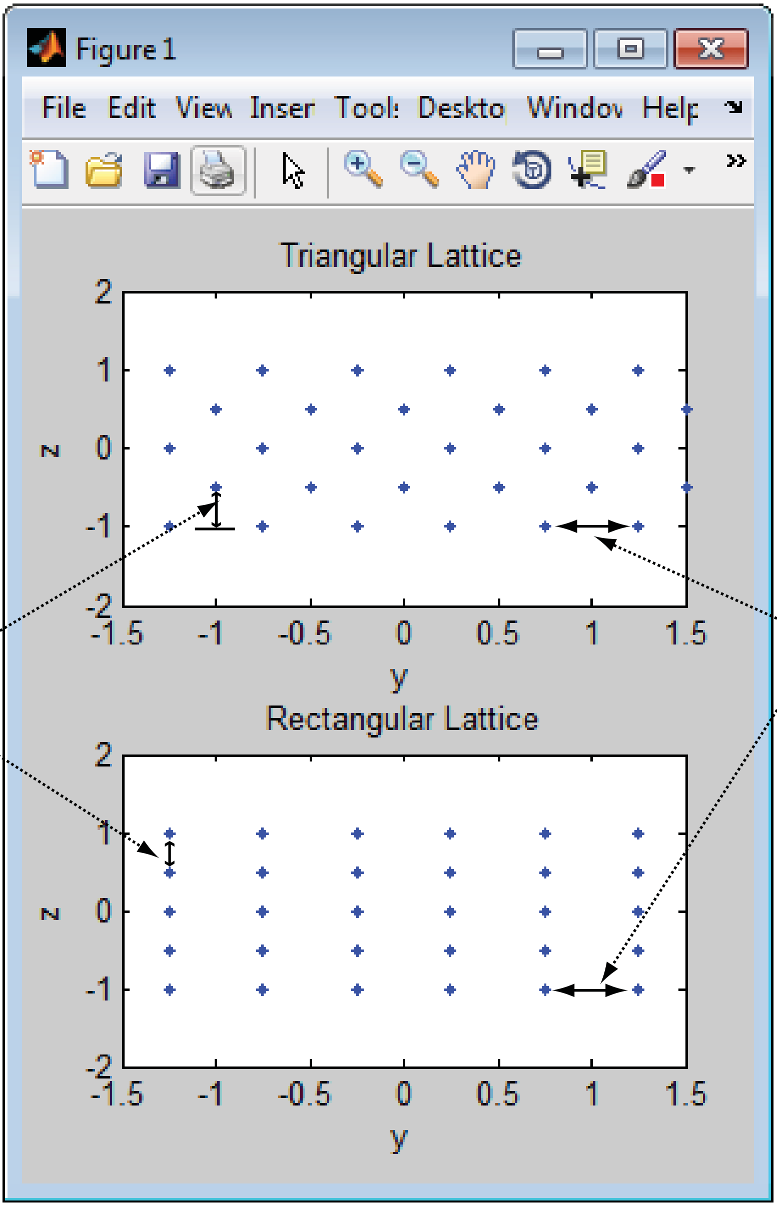


Figure 1
File Edit View Insert Tools Desktop Window Help
Triangular Lattice
2
1
0
-1
-2
z
-1.5
-1
-0.5
0
0.5
1
1.5
y
Rectangular Lattice
2
1
0
-1
-2
z
-1.5
-1
-0.5
0
0.5
1
1.5
y

Spacing Between
Rows

Examples

Azimuth Response of a 3-by-2 Heterogeneous URA

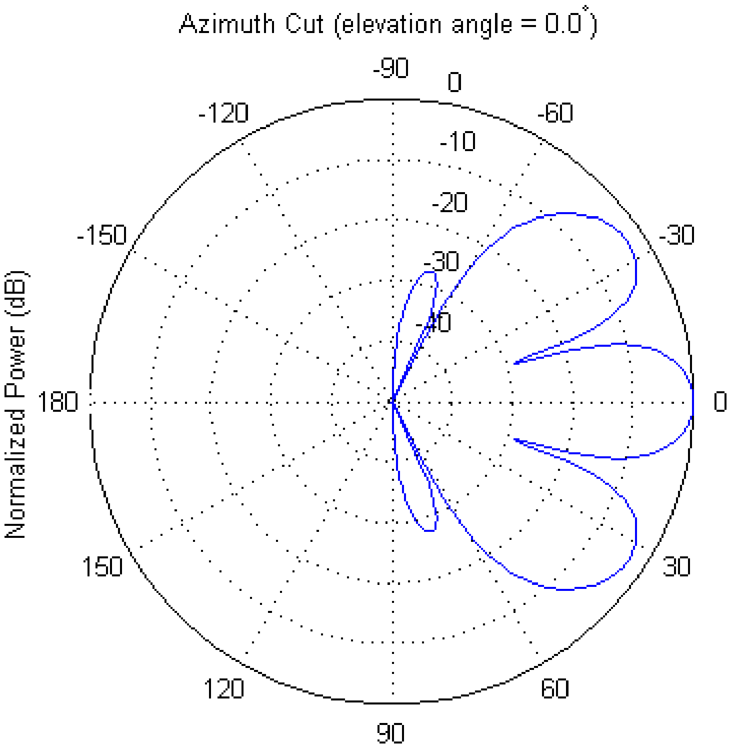
Construct a 3-by-2 heterogeneous URA with a rectangular lattice, and find the response of each element at 30 degrees azimuth and 0 degrees elevation. Assume the operating frequency is 1 GHz.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1; 2 2; 1 1]);
fc = 1e9;
ang = [30;0];
resp = step(sArray,fc,ang)
```

```
resp =

    0.8059
    0.7719
    0.8059
    0.8059
    0.7719
    0.8059
```

Plot the azimuth response of the array.

```
c = physconst('LightSpeed');
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar');
```

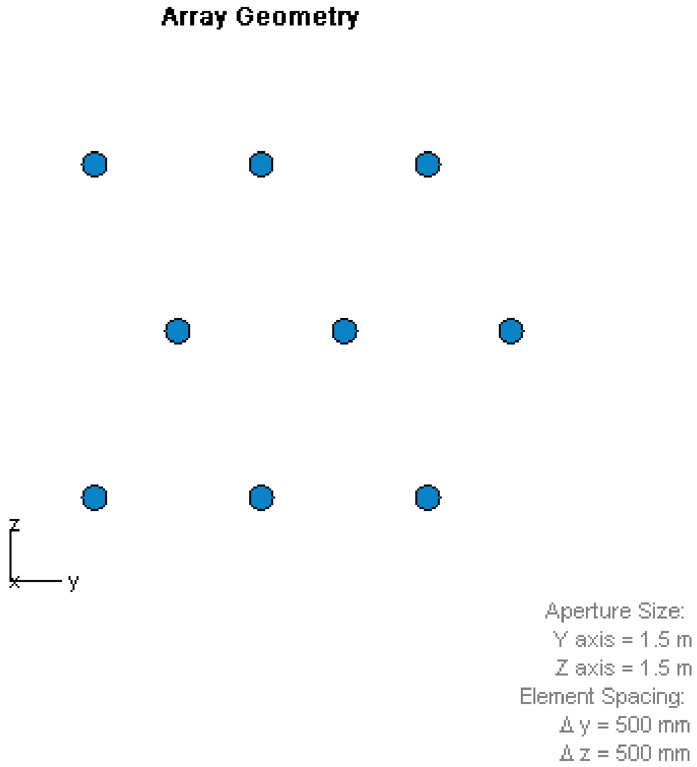


Normalized Power (dB), Broadside at 0.00 degrees

Draw Heterogeneous Triangular Lattice Array

Construct a 3-by-3 heterogeneous URA with a triangular lattice. The element spacing is 0.5 meter. Display the array shape.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1],...
    'Lattice','Triangular');
viewArray(sArray);
```


Array Geometry
z
x
y
Aperture Size:
Y axis = 1.5 m
Z axis = 1.5 m
Element Spacing:
Δ y = 500 mm
Δ z = 500 mm

References

[1] Brookner, E., ed. *Radar Technology*. Lexington, MA: LexBook, 1996.

[2] Brookner, E., ed. *Practical Phased Array Antenna Systems*. Boston: Artech House, 1991.

[3] Mailloux, R. J. “Phased Array Theory and Technology,” *Proceedings of the IEEE*, Vol., 70, Number 3, 1982, pp. 246–291.

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

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

See Also

`phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.URA` | `phased.HeterogeneousULA` | `phased.HeterogeneousConformalArray`

Related Examples

- Phased Array Gallery

phased.HeterogeneousURA.clone

Purpose Create new system object with identical 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 2-by-2 element heterogeneous URA with different cosine antenna patterns. The signals arrive from 10° and 30° azimuth. Both signals have an elevation angle of 0° degrees.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2; 1 2]);
y = collectPlaneWave(sArray,randn(4,2),[10 30],1e8,...
    physconst('LightSpeed'));
```

```
y(:,1)

ans =

   4.2642 - 0.5130i
   2.6971 - 0.2353i
  -0.6539 - 0.0625i
   2.8244 - 0.2227i
```

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`

phased.HeterogeneousURA.getElementPosition

Purpose Positions of array elements

Syntax

```
POS = getElementPosition(H)
POS = getElementPosition(H,ELEIDX)
```

Description `POS = getElementPosition(H)` returns the element positions of the HeterogeneousURA System object, `H`. `POS` is a 3-by-N matrix where N is the number of elements in `H`. Each column of `POS` defines the position of an element in the local coordinate system, in meters, using the form `[x; y; z]`.

For details regarding the local coordinate system of the URA or heterogeneous URA, enter `phased.URA.coordinateSystemInfo`.

`POS = getElementPosition(H,ELEIDX)` returns the positions of the elements that are specified in the element index vector, `ELEIDX`. The element indices of a URA run down each column, then to the top of the next column to the right. For example, in a URA with 4 elements in each row and 3 elements in each column, the element in the third row and second column has an index value of 6. This syntax can use any of the input arguments in the previous syntax.

Examples

Element Positions of Heterogeneous URA

Construct a heterogeneous URA with a rectangular lattice, and obtain the element positions.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2; 2 1]);
pos = getElementPosition(sArray);
pos =

         0         0         0         0
   -0.2500   -0.2500    0.2500    0.2500
    0.2500   -0.2500    0.2500   -0.2500
```

phased.HeterogeneousURA.getNumElements

Purpose Number of elements in array

Syntax `N = getNumElements(H)`

Description `N = getNumElements(H)` returns the number of elements, N, in the Heterogeneous URA object `H`.

Examples Construct a Heterogeneous URA, and obtain the number of elements.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2; 2 1]);
N = getNumElements(sArray)
```

```
N =

     4
```

phased.HeterogeneousURA.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.HeterogeneousURA.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.HeterogeneousURA.getTaper

Purpose

Array element tapers

Syntax

`wts = getTaper(h)`

Description

`wts = getTaper(h)` returns the tapers, `wts`, applied to each element of the phased heterogeneous uniform rectangular array (URA), `h`. Tapers are often referred to as weights.

Input Arguments

h - Uniform rectangular array

`phased.HeterogeneousURA` System object

Uniform rectangular array specified as a `phased.HeterogeneousURA` System object.

Output Arguments

wts - Array element tapers

N-by-1 complex-valued vector

Array element tapers returned as an N-by-1, complex-valued vector. The dimension N is the number of elements in the array. The array tapers are returned in the same order as the element indices. The element indices of a URA run down each column, then to the top of the next column to the right.

Examples

Heterogeneous URA Array Element Tapering

Construct a 2-by-5 element heterogeneous URA with a Taylor window taper along each row. Then, show the array with the element taper shading.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1 ; 1 2 2 2 1],...
    'Taper',[taylorwin(5)';taylorwin(5)']);
w = getTaper(sArray)

w =

    0.5181
    0.5181
    1.2029
    1.2029
    1.5581
    1.5581
    1.2029
    1.2029
    0.5181
    0.5181

viewArray(sArray,'ShowTaper',true);
```

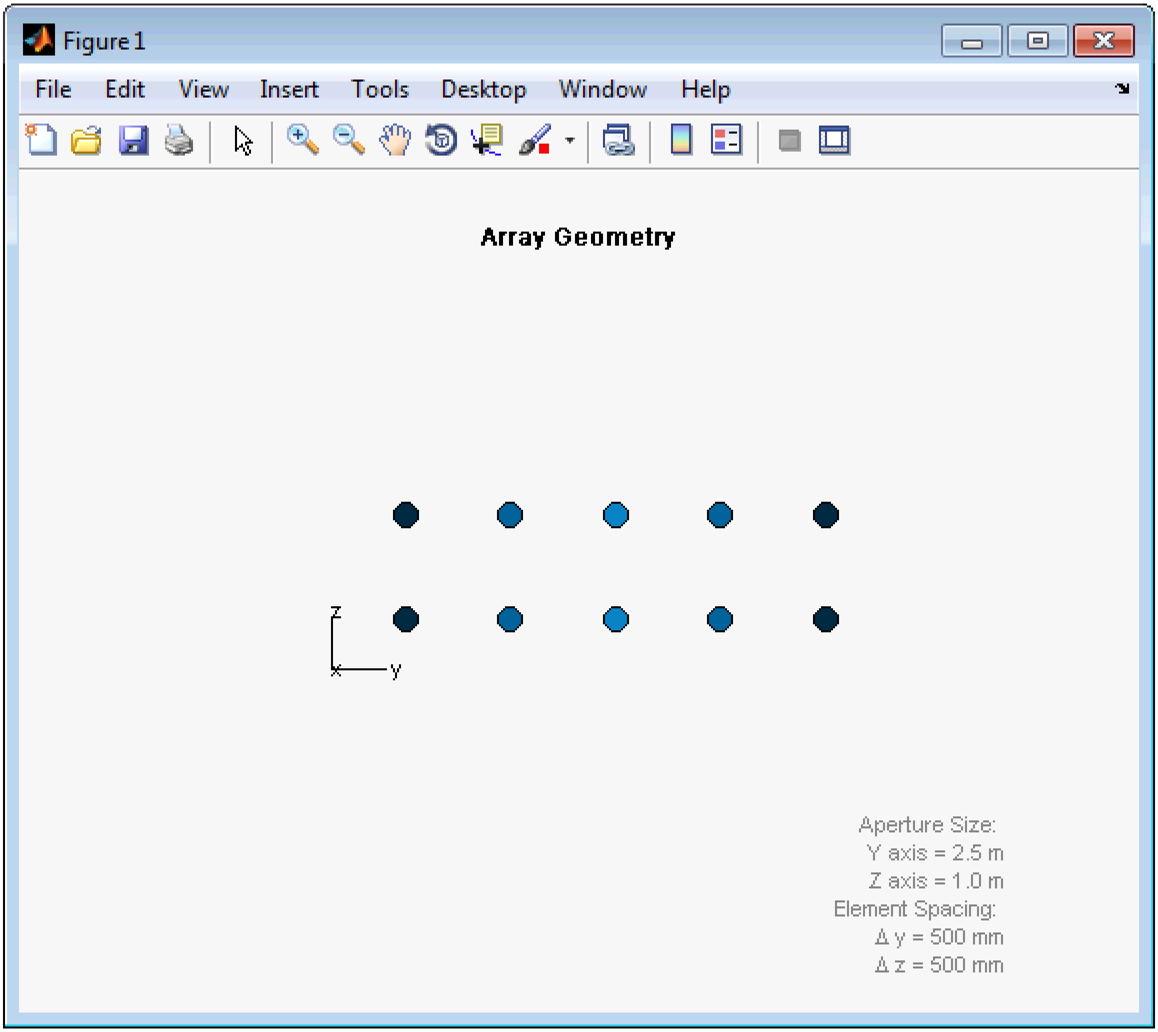


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Array Geometry
z
x
y
Aperture Size:
Y axis = 2.5 m
Z axis = 1.0 m
Element Spacing:
Δ y = 500 mm
Δ z = 500 mm

phased.HeterogeneousURA.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 `HeterogeneousURA` 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.HeterogeneousURA.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h - Uniform rectangular array

Uniform rectangular array specified as `phased.HeterogeneousURA` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability flag returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

Short-dipole Antenna Array Polarization

Show that an array of `phased.ShortDipoleAntennaElement` short-dipole antenna element supports polarization.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2 2 2 1 ; 1 2 2 2 1]);
isPolarizationCapable(sArray)
```

```
ans =

     1
```

The returned value `true (1)` shows that this array supports polarization.

phased.HeterogeneousURA.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 specified as a scalar or 1-by-K row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the array response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'Weights'

Weight values applied to the array, specified as a length-N column vector or N-by-M matrix. The dimension N is the number of elements in the array. The interpretation of M depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-M row vector	Apply one set of weights for the same single frequency or all M frequencies.
N-by-M matrix	Scalar	Apply all of the M different columns in `Weights` for the same single frequency.
	1-by-M row vector	Apply each of the M different columns in `Weights` for the corresponding frequency in `FREQ`.

'AzimuthAngles'

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the V coordinates for visualizing the radiation pattern in U/V space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

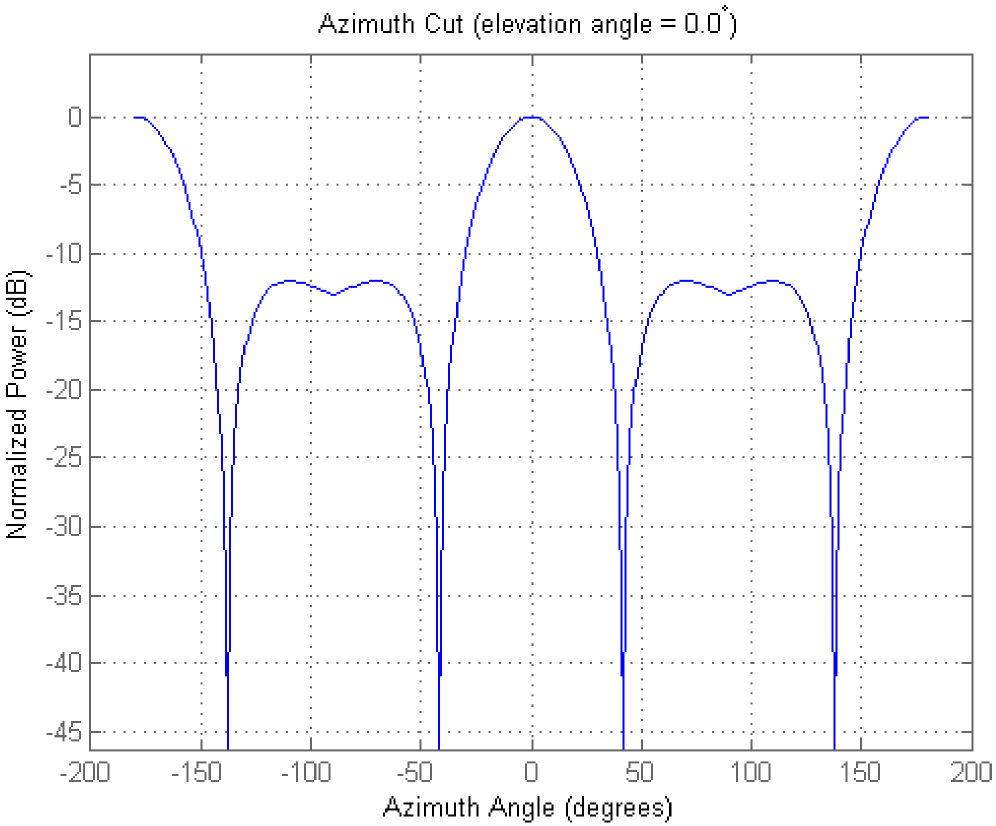
Default: `[-1:0.01:1]`

Examples

Azimuth Response and Directivity of Heterogeneous URA

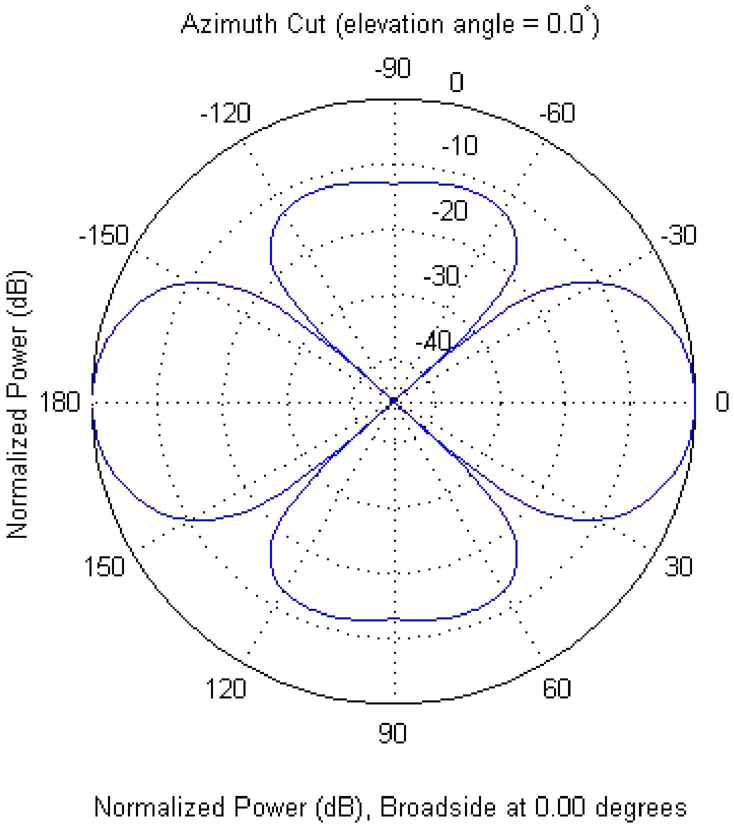
Construct a 3-by-3 heterogeneous URA with a rectangular lattice, then plot the array's azimuth response at 300 MHz.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1]);
fc = [3e8];
c = physconst('LightSpeed');
plotResponse(sArray,fc,c);
```



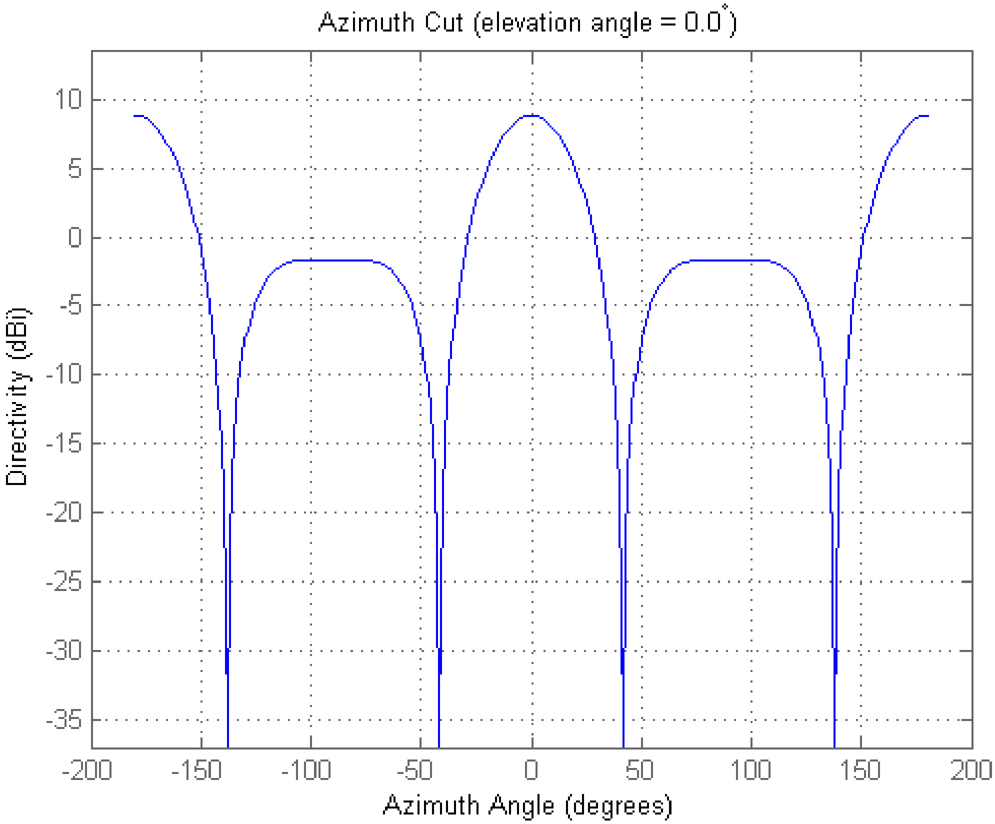
Plot the same result in polar form.

```
plotResponse(sArray,fc,c,'RespCut','Az','Format','Polar');
```



Finally, plot the directivity.

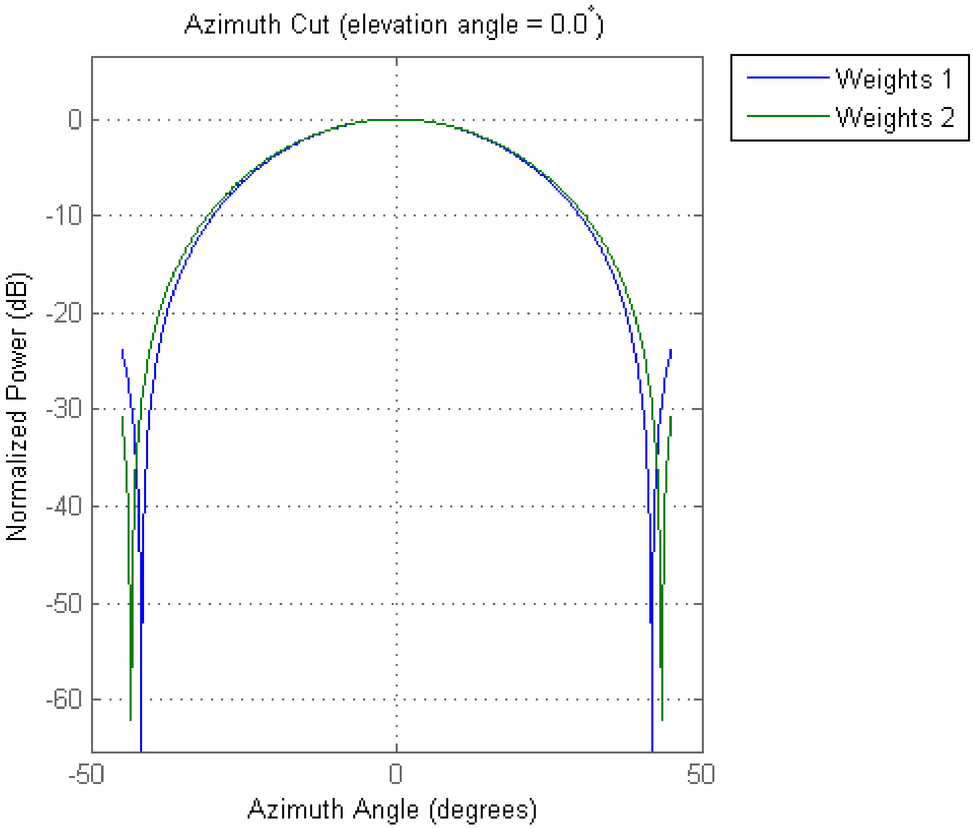
```
plotResponse(sArray,fc,c,'RespCut','Az','Unit','dbi');
```



Azimuth Responses of a Heterogeneous URA For Two Sets of Weights

Construct a square 3-by-3 heterogeneous URA composed of 9 short-dipole antenna elements with different orientations. Using the AzimuthAngles parameter, plot the array's azimuth response in the -45 degrees to 45 degrees in 0.1 degree increments. The `Weights` parameter lets you display the array's response simultaneously for different sets of weights: in this case a uniform set of weights and a tapered set.

```
sElement1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Z');
sElement2 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[2e8 5e8],...
    'AxisDirection','Y');
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1; 2 2 2; 1 1 1]);
fc = [3e8];
c = physconst('LightSpeed');
wts1 = ones(9,1)/9;
wts2 = [.7,.7,.7,.7,1,.7,.7,.7,.7]';
wts2 = wts2/sum(wts2);
plotResponse(sArray,fc,c,'RespCut','Az',...
    'Format','Line',...
    'AzimuthAngles',[-45:0.1:45],...
    'Weights',[wts1,wts2],'Unit','db');
```



See Also `uv2azel` | `azel2uv`

phased.HeterogeneousURA.release

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.

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

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions N-by-M-by-L. N is the number of elements in the array. The dimension M is the number of angles specified in `ANG`. L is the number of frequencies specified in `FREQ`. For any element, the columns of `RESP` contain the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.
- If the array is capable of supporting polarization, the voltage response, `RESP`, is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions N-by-M-by-L. N is the number of elements in the array, and M is the number of angles specified in `ANG`. L is the number of frequencies specified in `FREQ`. Each column of `RESP` contains the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.

Examples

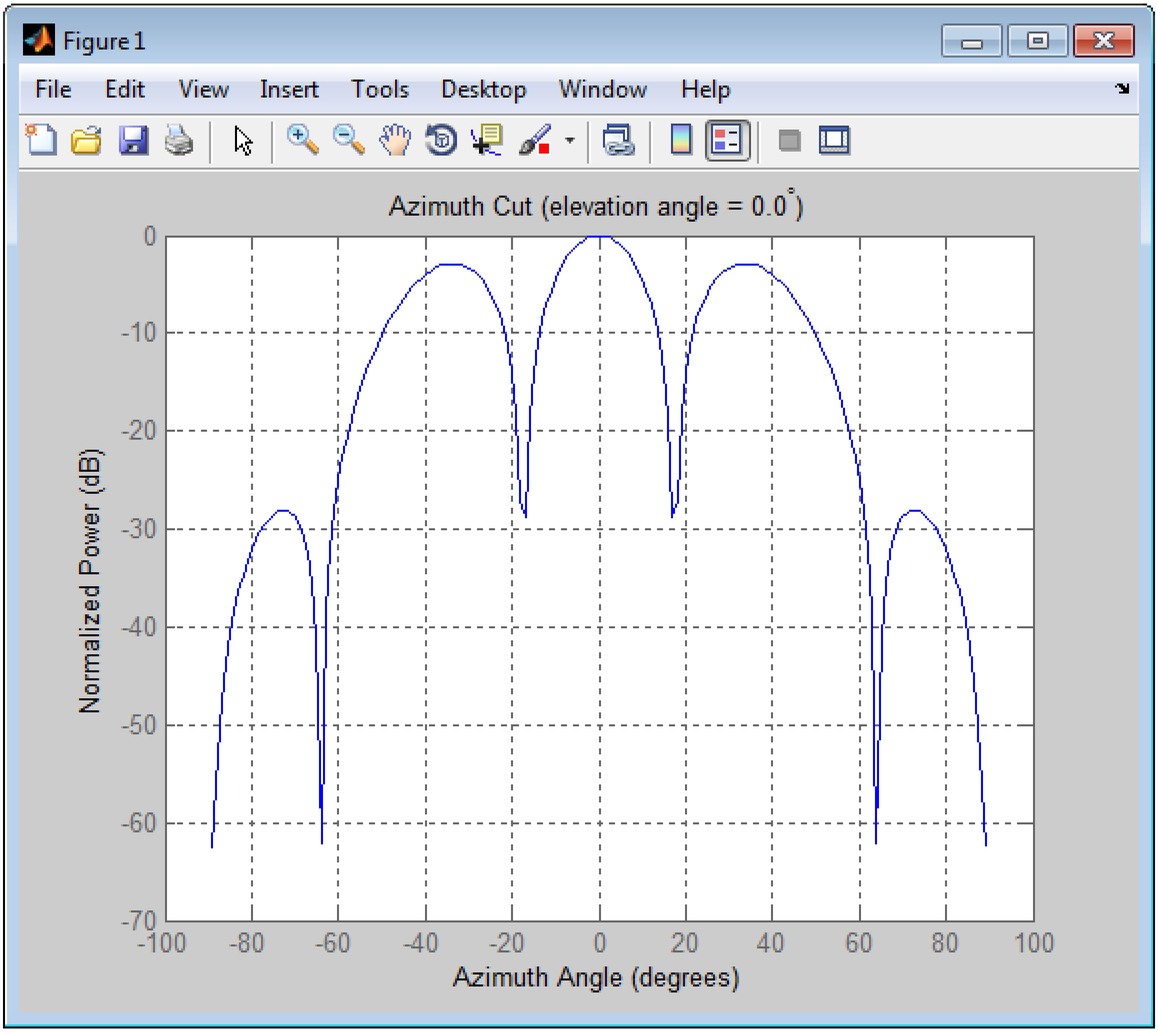
Response of a 2-by-2 Heterogeneous URA of Cosine Antennas

Construct a 2-by-2 rectangular lattice heterogeneous URA of cosine antenna elements, and find and plot the response of each element at 30° azimuth and 0° elevation. Assume the operating frequency is 1 GHz.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 2; 2 1]);
fc = 1e9;
c = physconst('LightSpeed');
ang = [30;0];
resp = step(sArray,fc,ang)
```

```
resp =

    0.8059
    0.7719
    0.7719
    0.8059
```

```
plotResponse(sArray,fc,c);
```



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. `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`

'ShowTaper'

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

'Title'

String specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

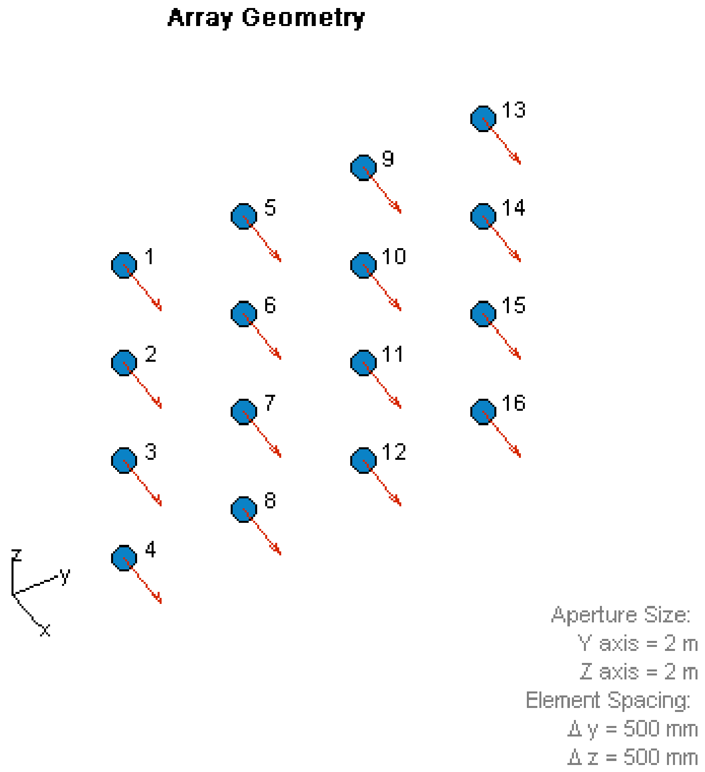
Handle of array elements in figure window.

Examples

Geometry, Normal Directions, and Indices of Heterogeneous URA Elements

Display the element positions, normal directions, and indices for all elements of a 4-by-4 heterogeneous URA.

```
sElement1 = phased.CosineAntennaElement('CosinePower',1.5);
sElement2 = phased.CosineAntennaElement('CosinePower',1.8);
sArray = phased.HeterogeneousURA(...
    'ElementSet',{sElement1,sElement2},...
    'ElementIndices',[1 1 1 1; 1 2 2 1; 1 2 2 1; 1 1 1 1]);
viewArray(sArray,'ShowIndex','all','ShowNormal',true);
```



See Also

`phased.ArrayResponse`

Related Examples

- Phased Array Gallery

Purpose

Isotropic antenna element

Description

The `IsotropicAntennaElement` object creates an antenna element with an isotropic response pattern. This antenna object does not support polarization.

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

1. Define and set up your isotropic antenna element. See "Construction" on page 1-596.
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 Hz) as a 1-by-2 row vector in the form of `[LowerBound HigherBound]`. The antenna element has zero response outside the specified frequency range.

Default: `[0 1e20]`

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
isPolarizationCapable	Polarization capability
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');
```

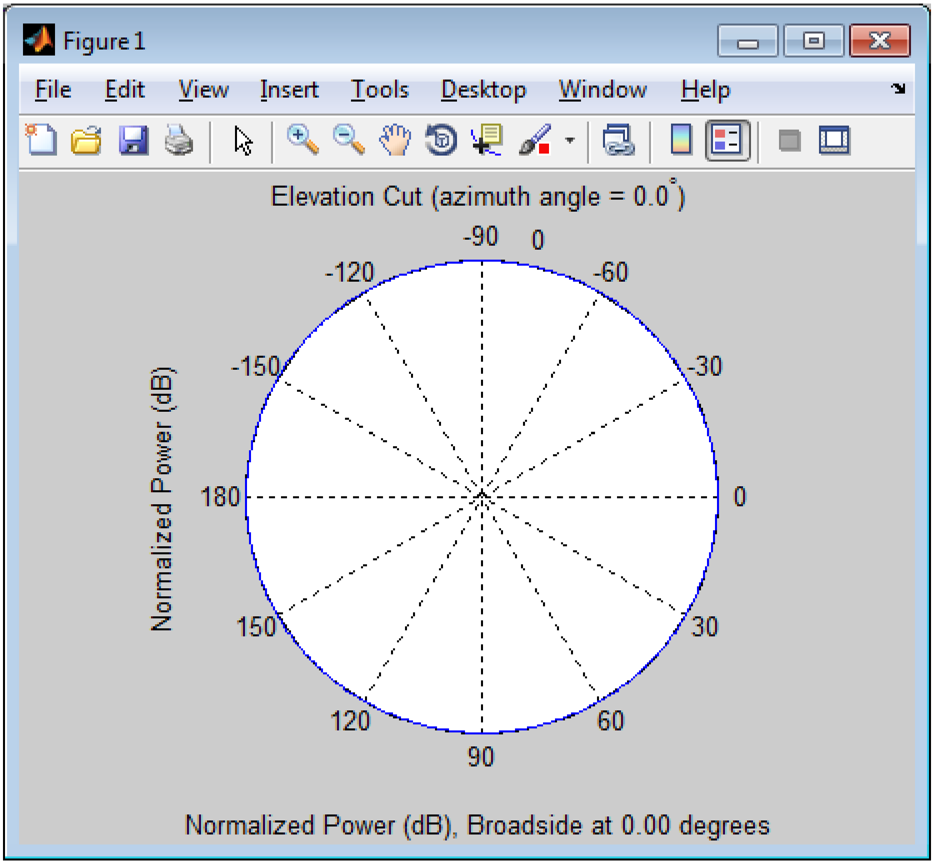


Figure 1
File Edit View Insert Tools Desktop Window Help
Elevation Cut (azimuth angle = 0.0°)
Normalized Power (dB)
-90
0
-120
-60
-150
-30
180
0
150
30
120
60
90
Normalized Power (dB), Broadside at 0.00 degrees

See Also

phased.ConformalArray | phased.CosineAntennaElement | phased.CrossedDipoleAntennaElement | phased.CustomAntennaElement | phased.CustomMicrophoneElement | phased.OmnidirectionalMicrophoneElement | phased.ShortDipoleAntennaElement | 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.

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.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.IsotropicAntennaElement` System object supports polarization. An antenna element supports polarization if it can create or respond to polarized fields. This object does not support polarization.

Input Arguments

h - Isotropic antenna element

Isotropic antenna element specified as a `phased.IsotropicAntennaElement` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not. Since the `phased.IsotropicAntennaElement` object does not support polarization, `flag` is always returned as `false`.

Examples

Isotropic Antenna Does Not Support Polarization

Determine whether a `phased.IsotropicAntennaElement` antenna element supports polarization.

```
h = phased.IsotropicAntennaElement('FrequencyRange',[1.0,10]*1e9);
isPolarizationCapable(h)
```

```
ans =

     0
```

The returned value `false (0)` shows that the antenna element does not support polarization.

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 System object

FREQ

Operating frequency in Hertz specified as a scalar or 1–by-K row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the antenna response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'AzimuthAngles'

Azimuth angles for plotting element response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting element response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting element response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting element response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

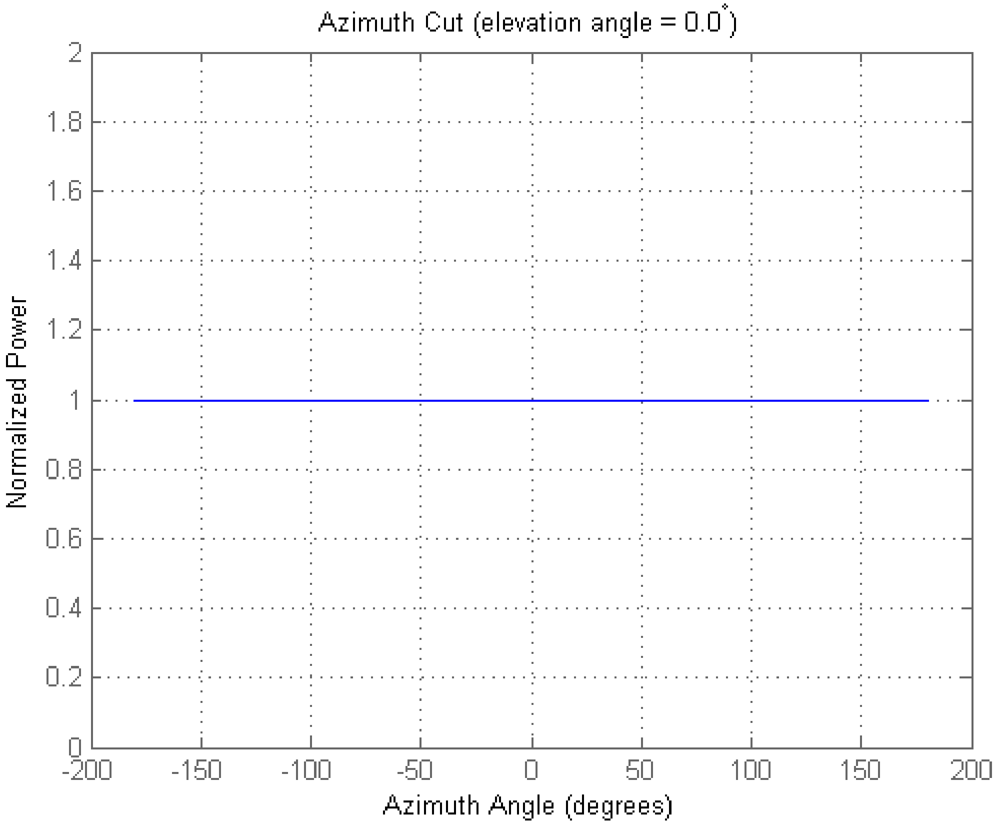
Examples

Plot Response and Directivity of Isotropic Antenna

This example shows how to plot the response and the directivity of an isotropic antenna element.

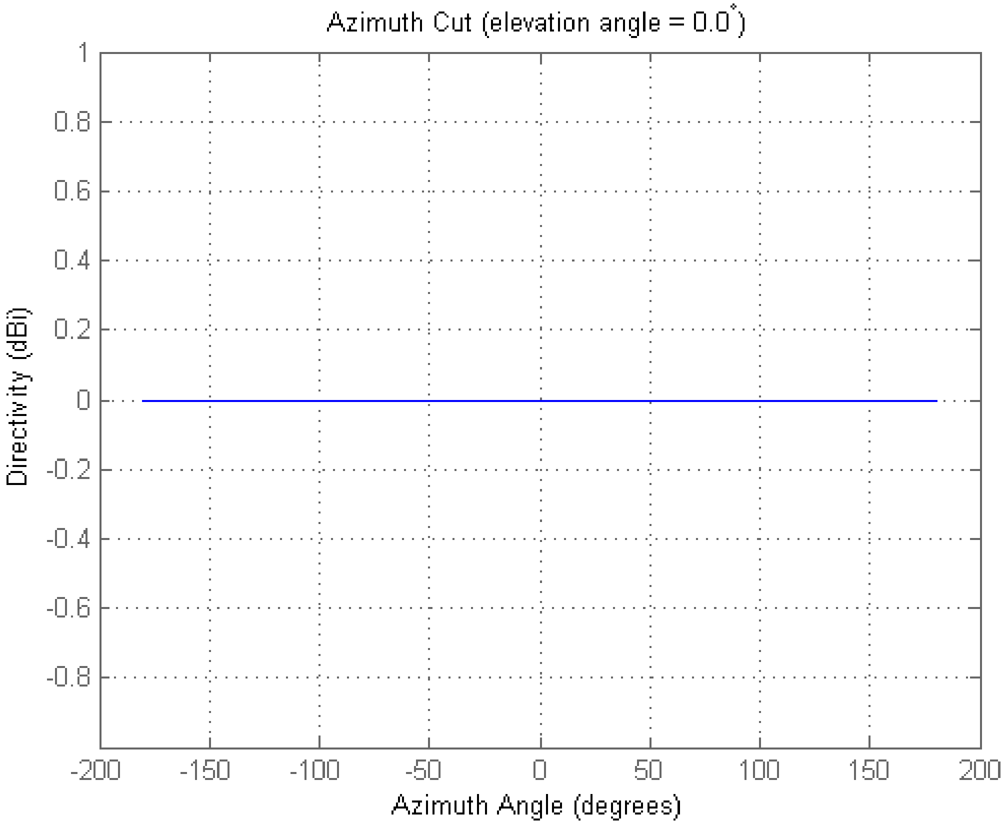
Draw a line plot of an azimuth cut of the response of an isotropic antenna along 0 degrees elevation. Assume the operating frequency is 1 GHz.

```
sIso = phased.IsotropicAntennaElement;
plotResponse(sIso,1e9,'Unit','pow');
```



Draw an azimuth cut of the antenna directivity.

```
plotResponse(sIso,1e9,'Unit','dbi');
```



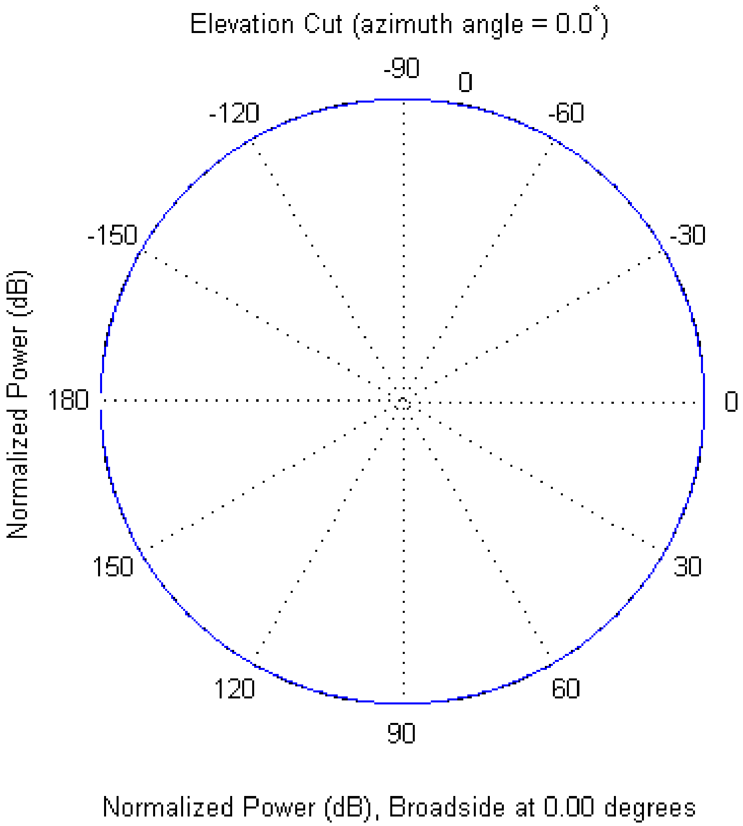
Plot Elevation-Cut of Isotropic Antenna Response

Construct an isotropic antenna operating in the frequency range from 800 MHz to 1.2 GHz. Find the response of the antenna at boresight at 1 GHz.

```
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[800e6 1.2e9]);
fc = 1e9;
resp = step(sIso,fc,[0;0])
resp =

     1
```

Plot the polar-form of the elevation response of the antenna.

```
plotResponse(sIso,fc,'RespCut','El','Format','Polar');
```



Plot 3-D Response

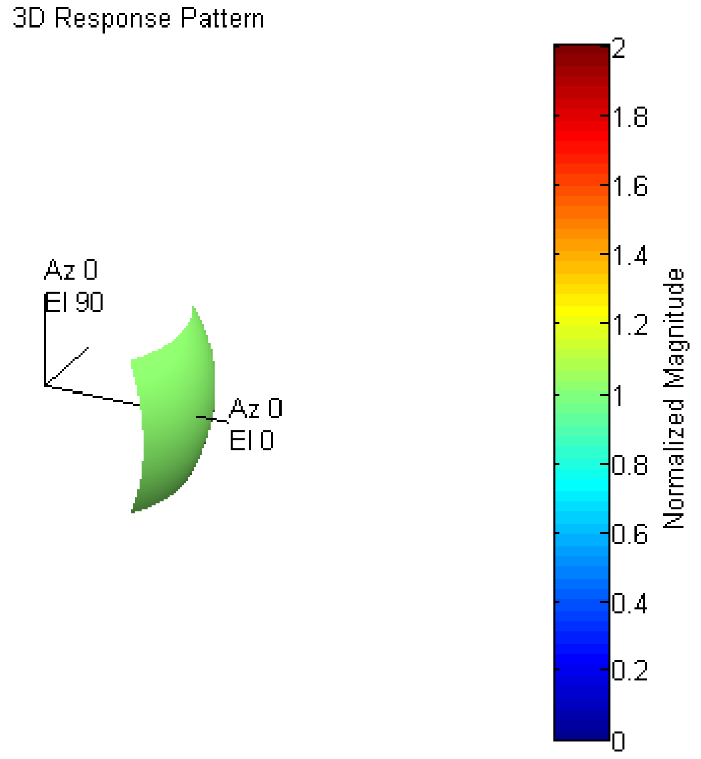
This example shows how to construct an isotropic antenna operating over a frequency range from 800 MHz to 1.2 GHz and how to plot its response.

Construct the antenna element.

```
sIso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[0.8e9 1.2e9]);
```

Plot the 3-D response of the antenna at 1 GHz from -30 to 30 degrees in both azimuth and elevation at 0.1 degree increments.

```
fc = 1e9;
plotResponse(sIso,fc,'RespCut','3D','Format','Polar',...
    'Unit','mag','AzimuthAngles',[-30:.1:30],...
    'ElevationAngles',[-30:.1:30]);
```



See Also `uv2azel` | `azel2uv`

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.

---

phased.IsotropicAntennaElement.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 specified as an *M*-by-*L*, complex-valued matrix. In this matrix, *M* represents the number of angles specified in `ANG` while *L* represents the number of frequencies specified in `FREQ`.

Examples

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');
```

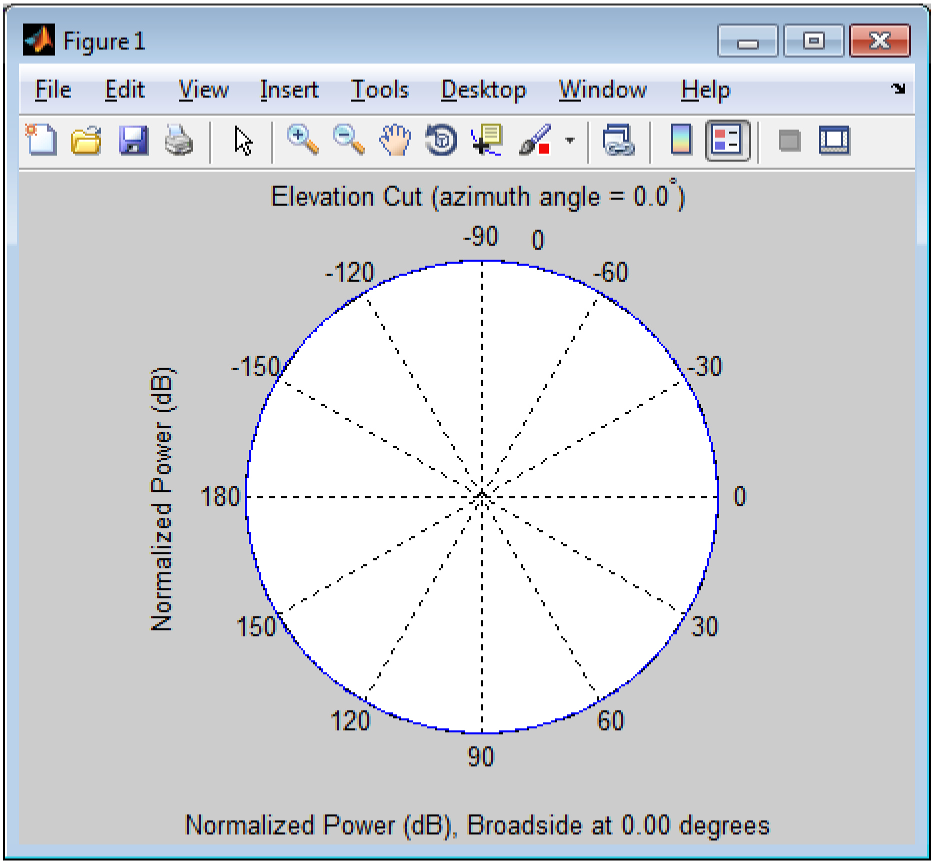


Figure 1
File Edit View Insert Tools Desktop Window Help
Elevation Cut (azimuth angle = 0.0°)
-90
0
-120
-60
-150
-30
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

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 1-619.
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');
```

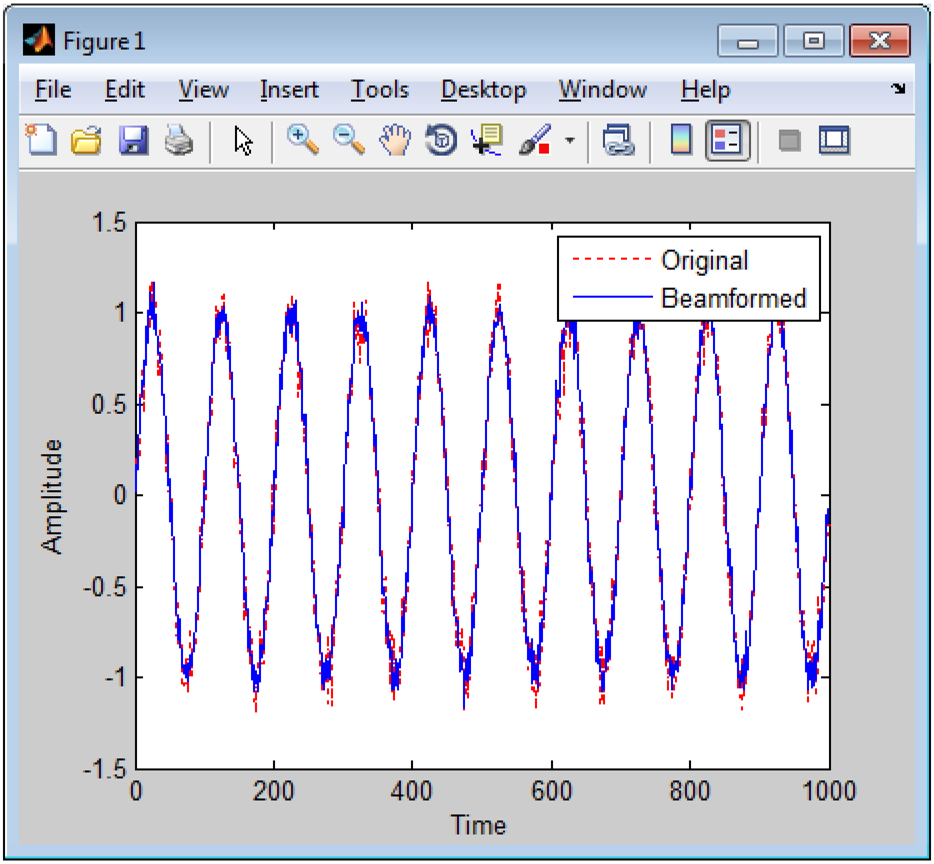


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
1.5
1
0.5
0
-0.5
-1
-1.5
Amplitude
0
200
400
600
800
1000
Time
Original
Beamformed

References

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

See Also

`phased.MVDRBeamformer` | `phased.PhaseShiftBeamformer` | `phased.TimeDelayLCMVBeamformer`

Concepts

- "Adaptive Beamforming"

phased.LCMVBeamformer.clone

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`.

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.LCMVBeamformer.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.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.

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 1-631.
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);
```

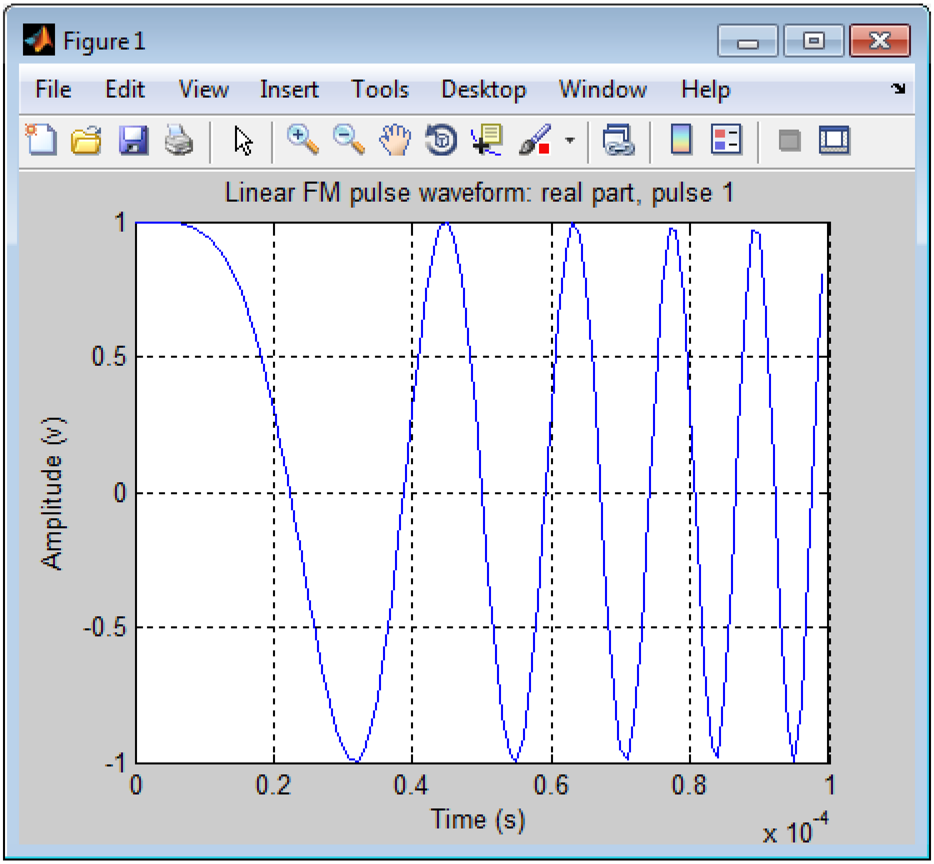


Figure 1
File Edit View Insert Tools Desktop Window Help
Linear FM pulse waveform: real part, pulse 1
1
0.5
0
-0.5
-1
Amplitude (v)
0
0.2
0.4
0.6
0.8
1
Time (s)
x 10^-4

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

phased.LinearFMWaveform.bandwidth

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)
```

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');
```

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.LinearFMWaveform.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

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.

```
[Pxx,F] = periodogram(y,[],2048,hs.SampleRate,'centered');
plot(F/1000,10*log10(Pxx)); grid;
xlabel('Frequency (kHz)');
ylabel('Power/Frequency (dB/Hz)');
title('Periodogram Power Spectrum Density Estimate');
```

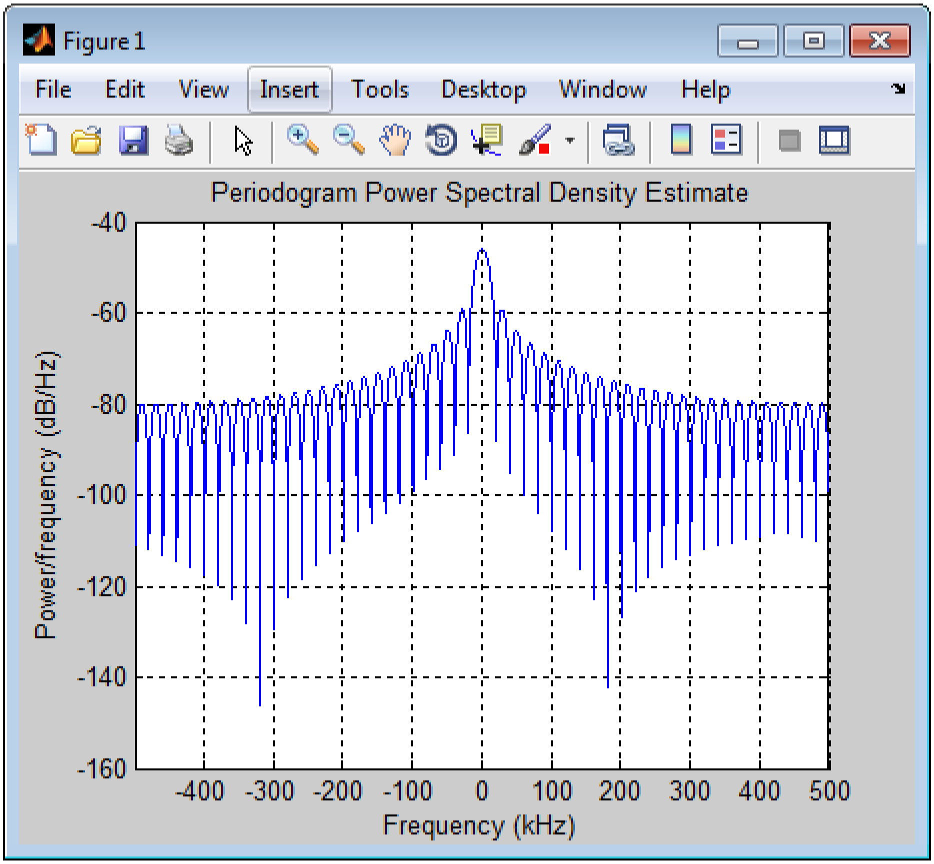


Figure 1
File Edit View Insert Tools Desktop Window Help
Periodogram Power Spectral Density Estimate
Power/frequency (dB/Hz)
Frequency (kHz)
-40
-60
-80
-100
-120
-140
-160
-400
-300
-200
-100
0
100
200
300
400
500

Detect the range.

```
[~,rngidx] = findpeaks(pow2db(Pxx/max(Pxx)),...
    'MinPeakHeight',-5);
rngfreq = F(rngidx);
re = stretchfreq2rng(rngfreq,hs.SweepSlope,...
    hs.ReferenceRange,c);
```

See Also

`phased.StretchProcessor` | `stretchfreq2rng`

Related Examples

- Range Estimation Using Stretch Processing

Concepts

- "Stretch Processing"

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 `PlotType` 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. `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);
```

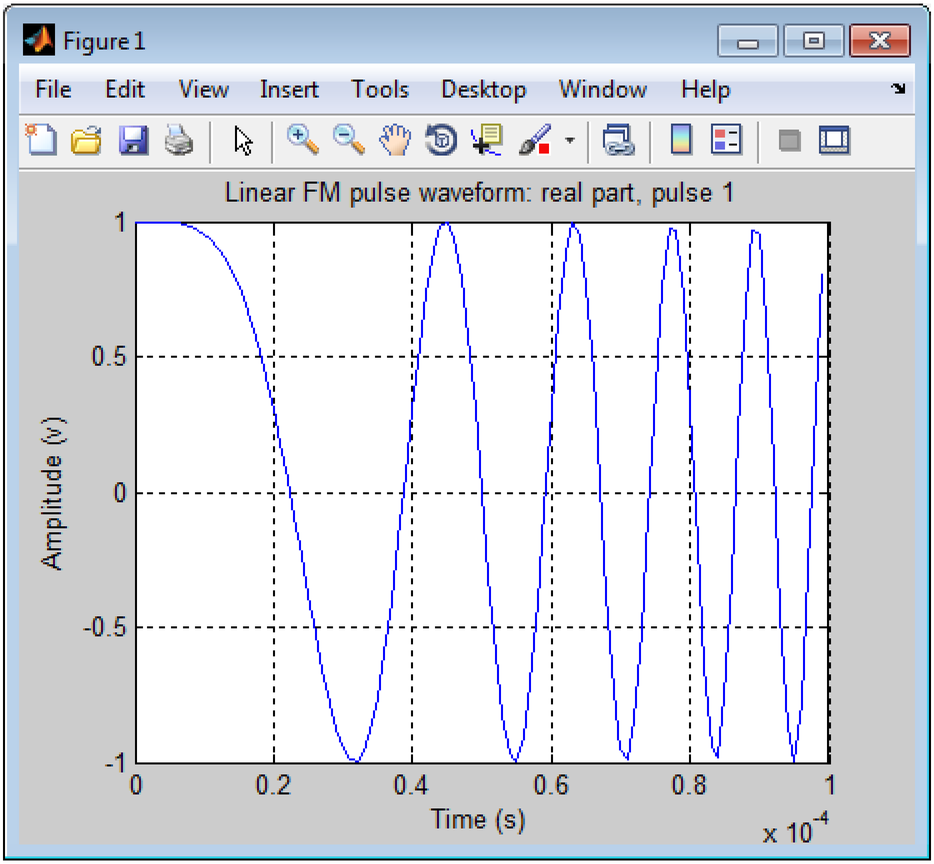


Figure 1
File Edit View Insert Tools Desktop Window Help
Linear FM pulse waveform: real part, pulse 1
Amplitude (v)
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1
Time (s)
x 10⁻⁴

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.

phased.LinearFMWaveform.reset

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.

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 1-654.
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');
```

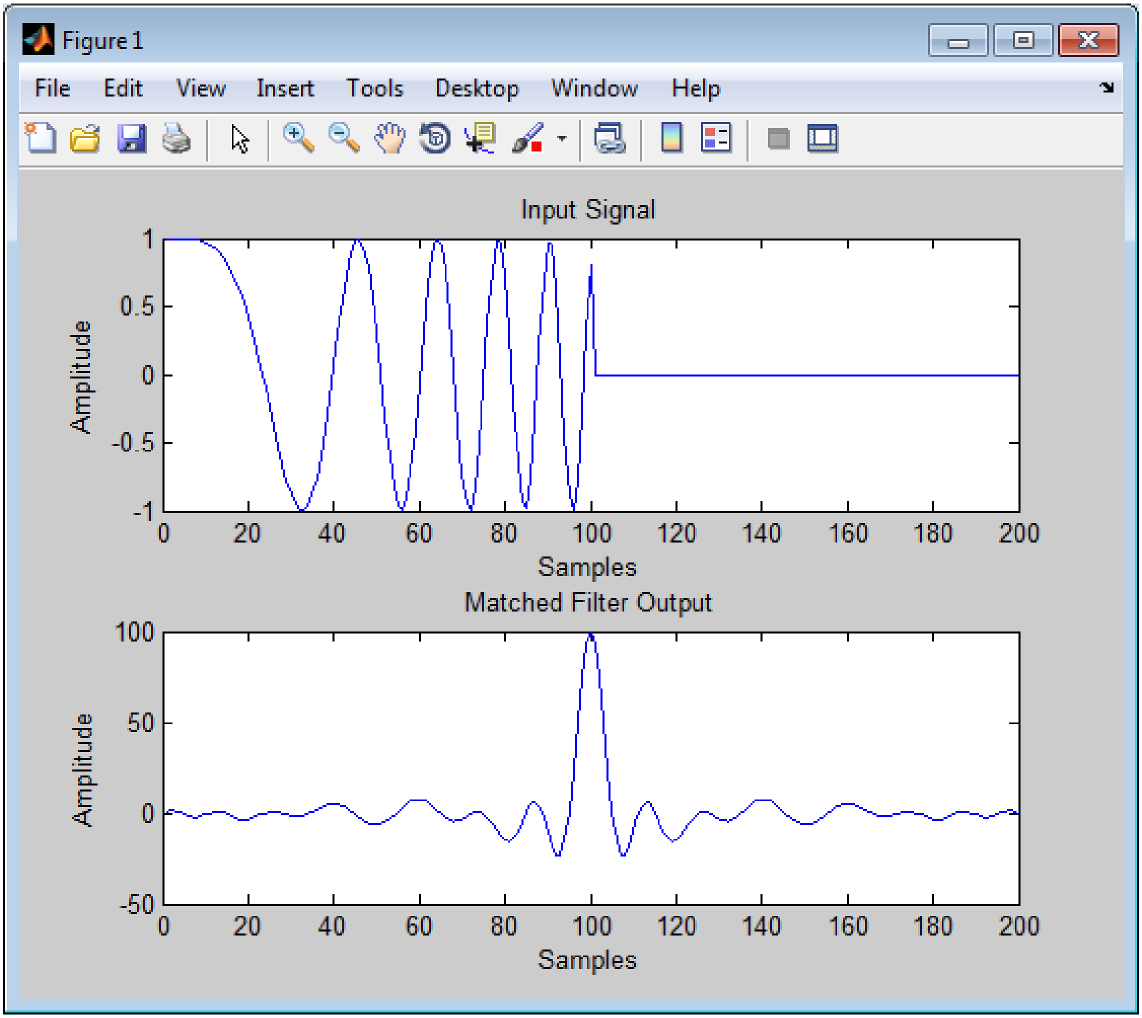


Figure 1
File Edit View Insert Tools Desktop Window Help
Input Signal
Amplitude
1
0.5
0
-0.5
-1
0 20 40 60 80 100 120 140 160 180 200
Samples
Matched Filter Output
Amplitude
100
50
0
-50
0 20 40 60 80 100 120 140 160 180 200
Samples

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');
```

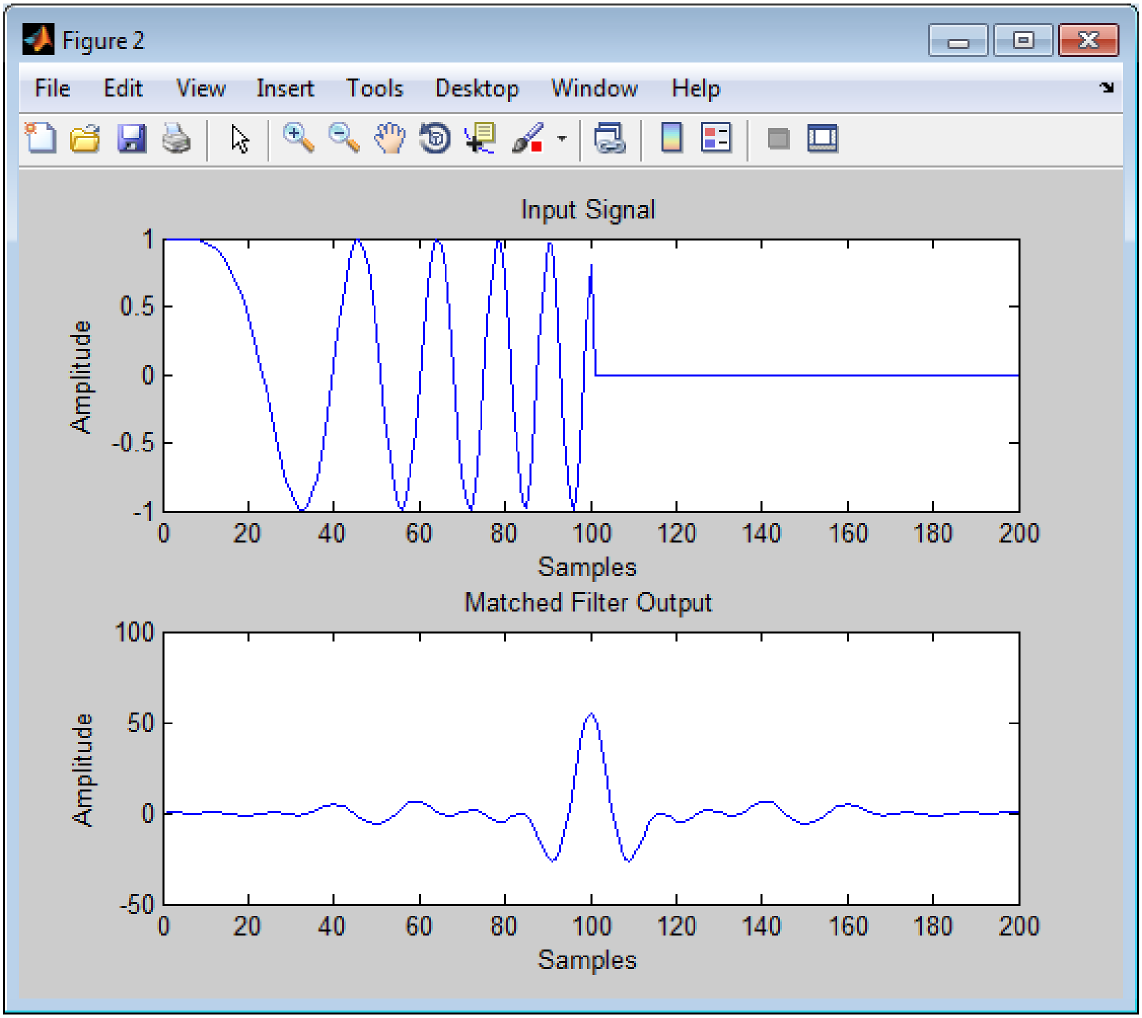


Figure 2
File Edit View Insert Tools Desktop Window Help
Input Signal
Amplitude
1
0.5
0
-0.5
-1
0 20 40 60 80 100 120 140 160 180 200
Samples
Matched Filter Output
Amplitude
100
50
0
-50
0 20 40 60 80 100 120 140 160 180 200
Samples

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');
```

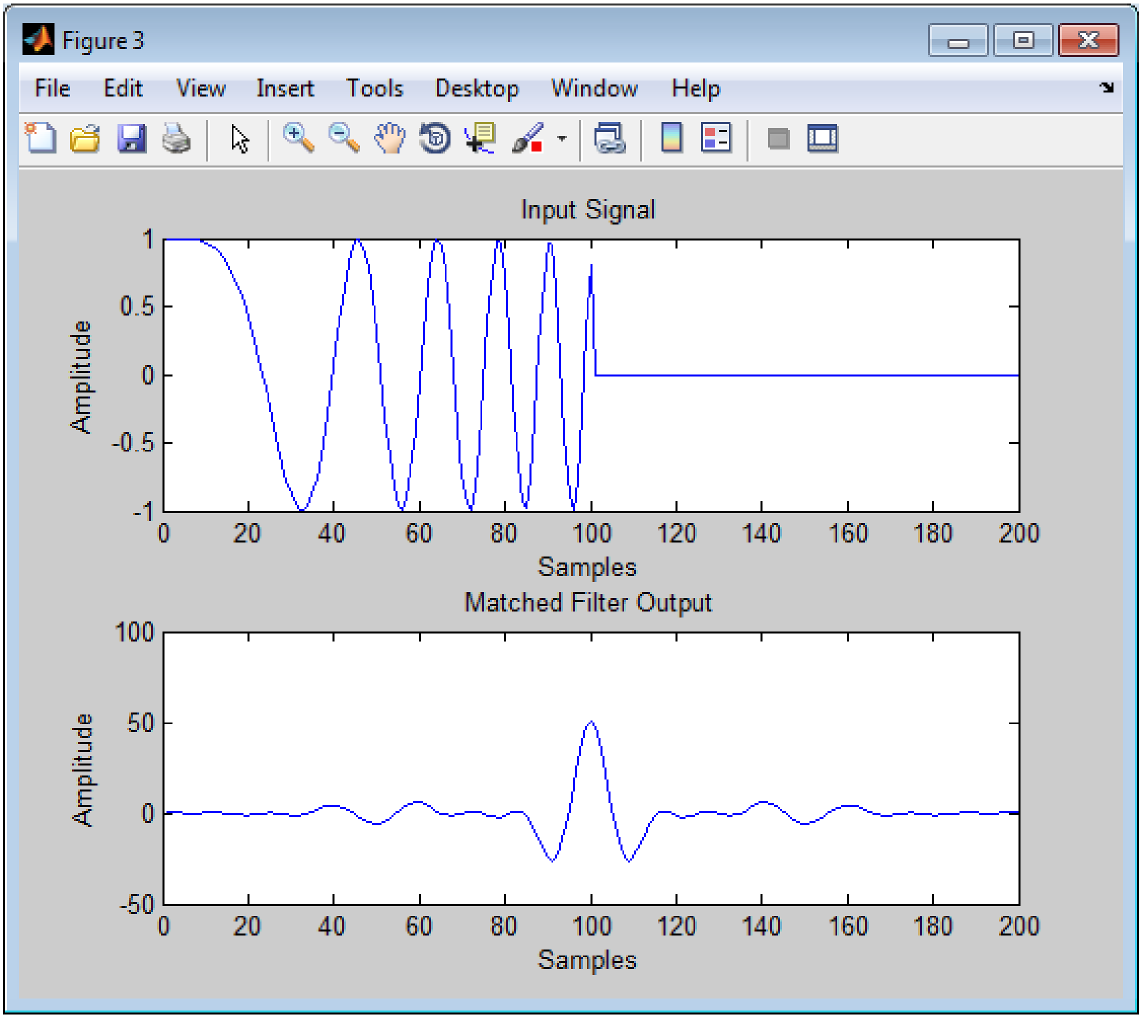


Figure 3
File Edit View Insert Tools Desktop Window Help
Input Signal
Amplitude
1
0.5
0
-0.5
-1
0 20 40 60 80 100 120 140 160 180 200
Samples
Matched Filter Output
Amplitude
100
50
0
-50
0 20 40 60 80 100 120 140 160 180 200
Samples

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` | `pulsint` | `phased.StretchProcessor` | `phased.TimeVaryingGain` | `taylorwin`

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`.

phased.MatchedFilter.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.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.

phased.MatchedFilter.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 `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.

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.MatchedFilter.step

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);
```

phased.MVDRBeamformer

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 1-672.

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

Sensor array

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

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

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);
```

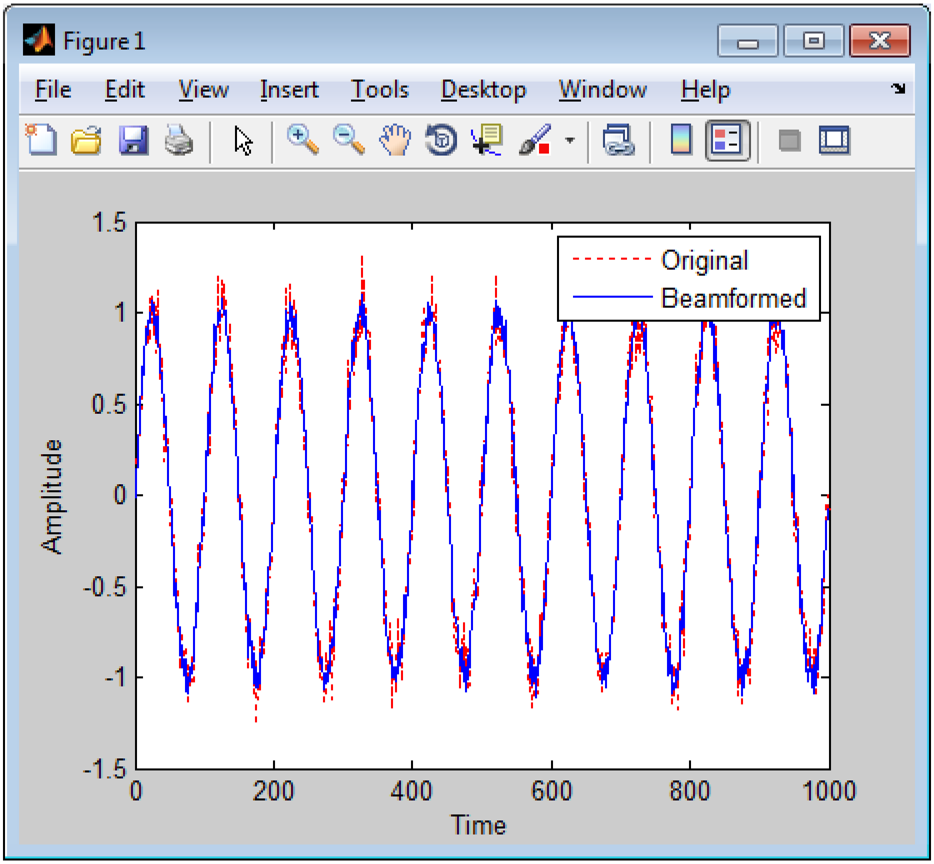


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Amplitude
Time
Original
Beamformed
1.5
1
0.5
0
-0.5
-1
-1.5
0
200
400
600
800
1000

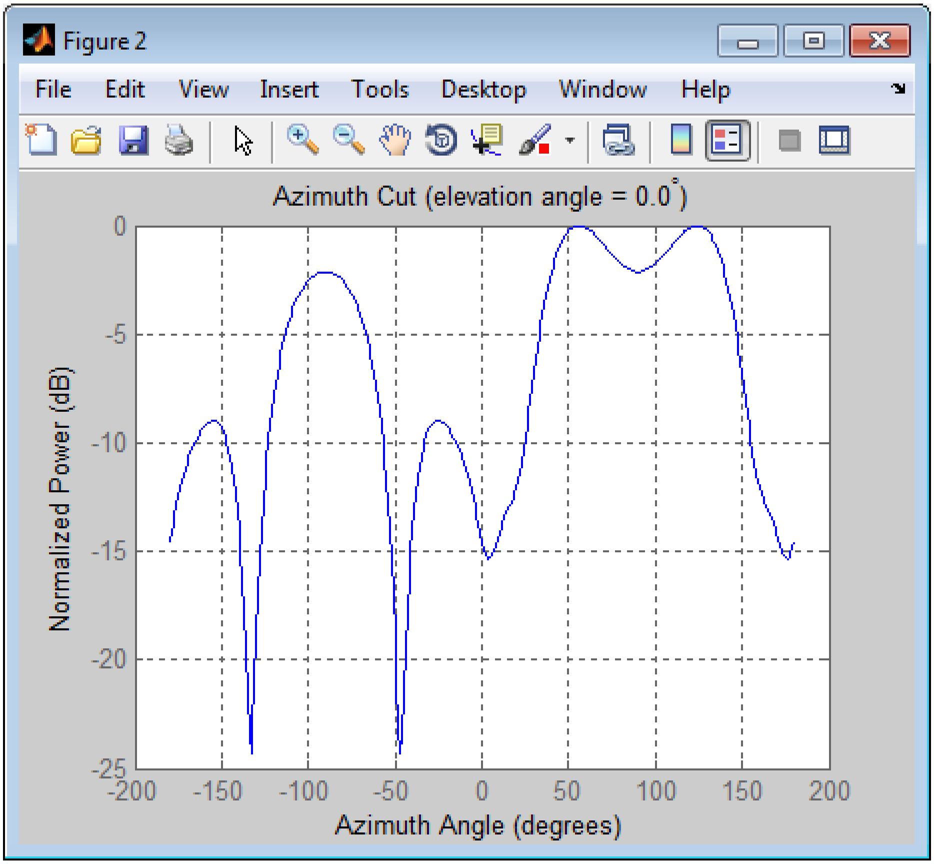


Figure 2
File
Edit
View
Insert
Tools
Desktop
Window
Help
Azimuth Cut (elevation angle = 0.0°)
Normalized Power (dB)
0
-5
-10
-15
-20
-25
-200
-150
-100
-50
0
50
100
150
200
Azimuth Angle (degrees)

References

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

See Also

`phased.FrostBeamformer` | `phased.PhaseShiftBeamformer` | `phased.LCMVBeamformer` | `uv2azel` | `phitheta2azel`

phased.MVDRBeamformer.clone

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`.

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.MVDRBeamformer.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 `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.

phased.MVDRBeamformer.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 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 1-686.
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);
```

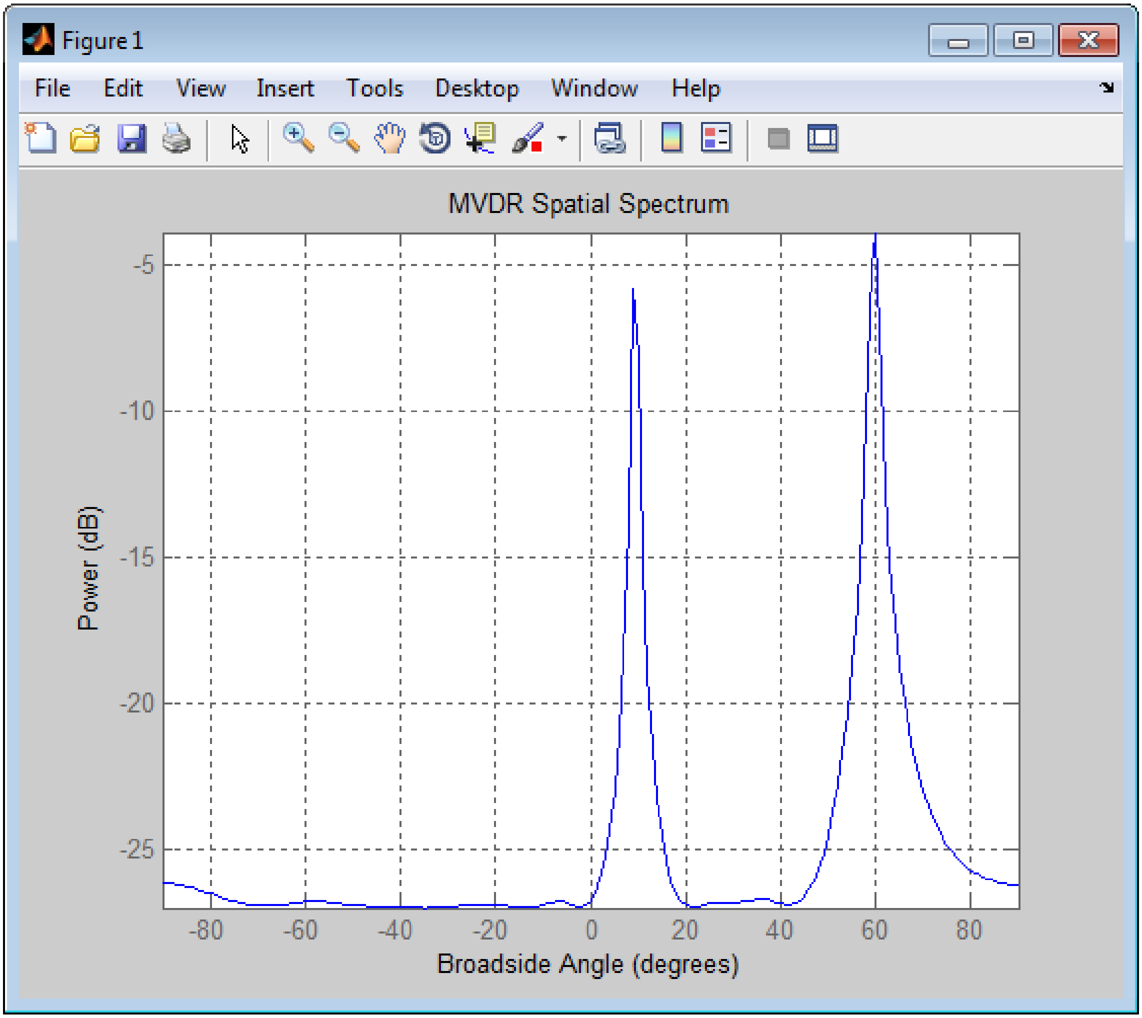


Figure 1
File Edit View Insert Tools Desktop Window Help
MVDR Spatial Spectrum
Power (dB)
-5
-10
-15
-20
-25
-80
-60
-40
-20
0
20
40
60
80
Broadside Angle (degrees)

References

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

See Also

`broadside2az` | `phased.MVDREstimator2D`

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.

phased.MVDREstimator.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 `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.

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. `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);
```

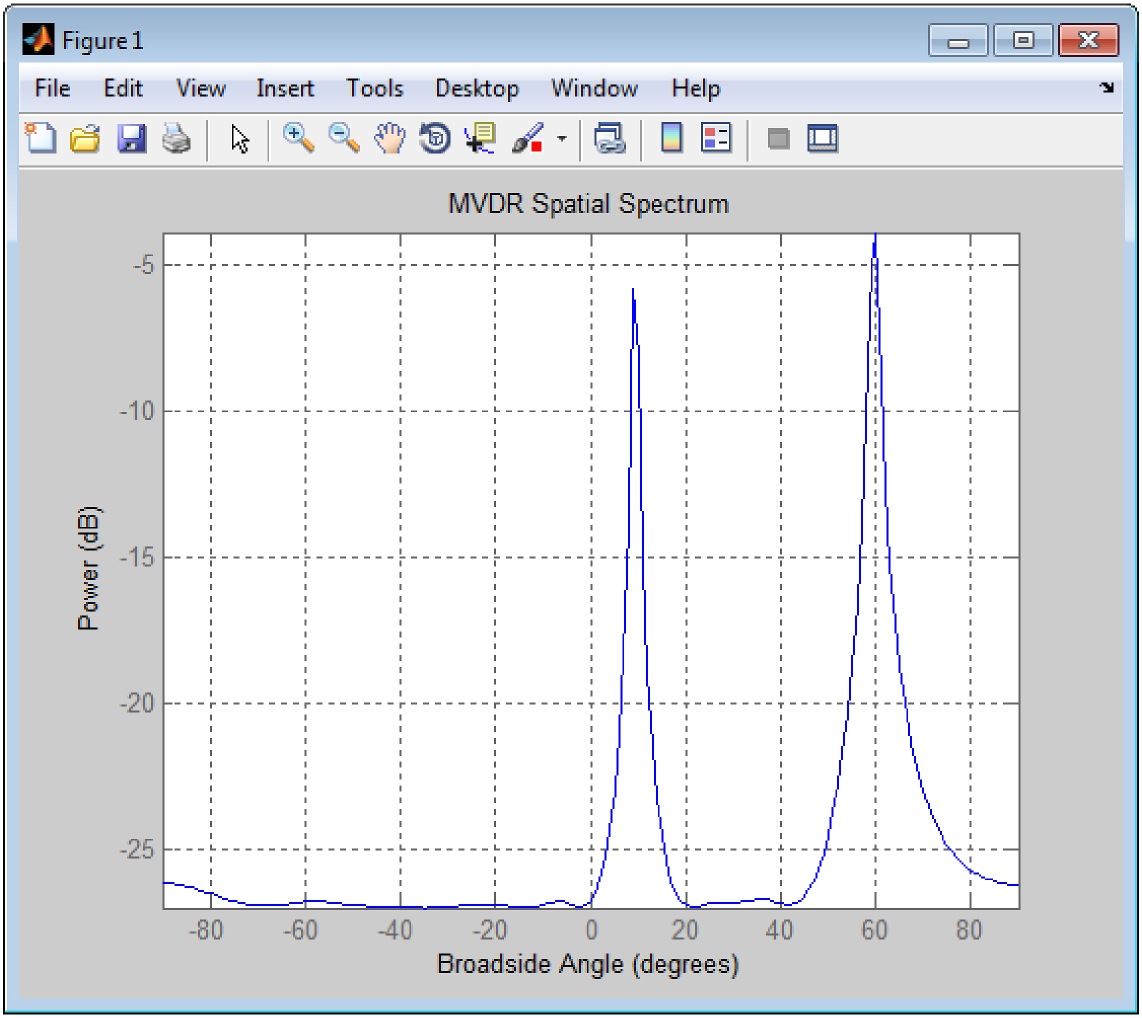


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
MVDR Spatial Spectrum
Power (dB)
-5
-10
-15
-20
-25
-80
-60
-40
-20
0
20
40
60
80
Broadside Angle (degrees)

phased.MVDREstimator.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 MVDR spatial spectrum estimator object

Syntax `reset(H)`

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

phased.MVDREstimator.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 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 1-702.
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

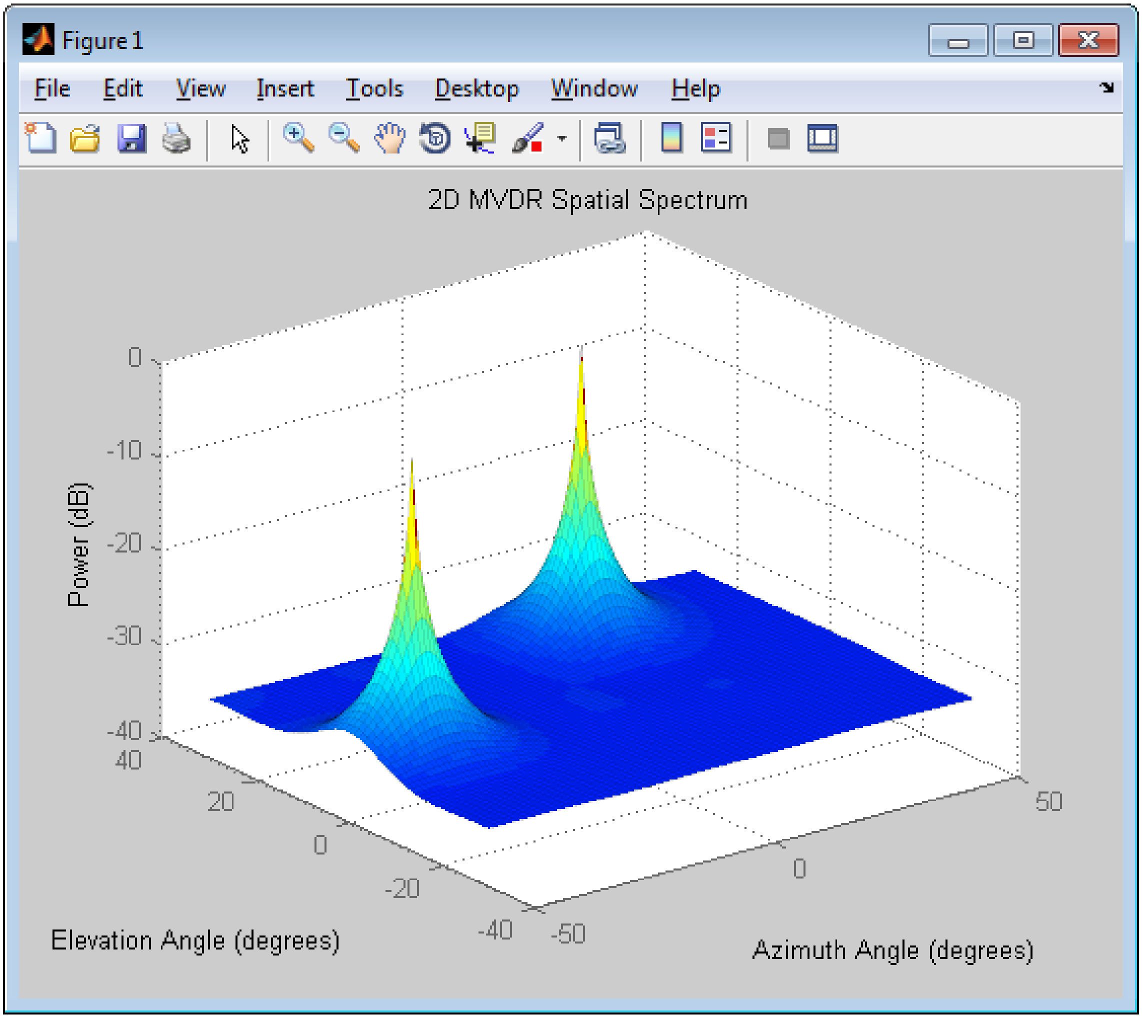
clone	Create 2-D MVDR spatial spectrum estimator object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
plotSpectrum	Plot spatial spectrum
release	Allow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

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`.

phased.MVDREstimator2D.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.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.

phased.MVDREstimator2D.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 `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. `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);
```

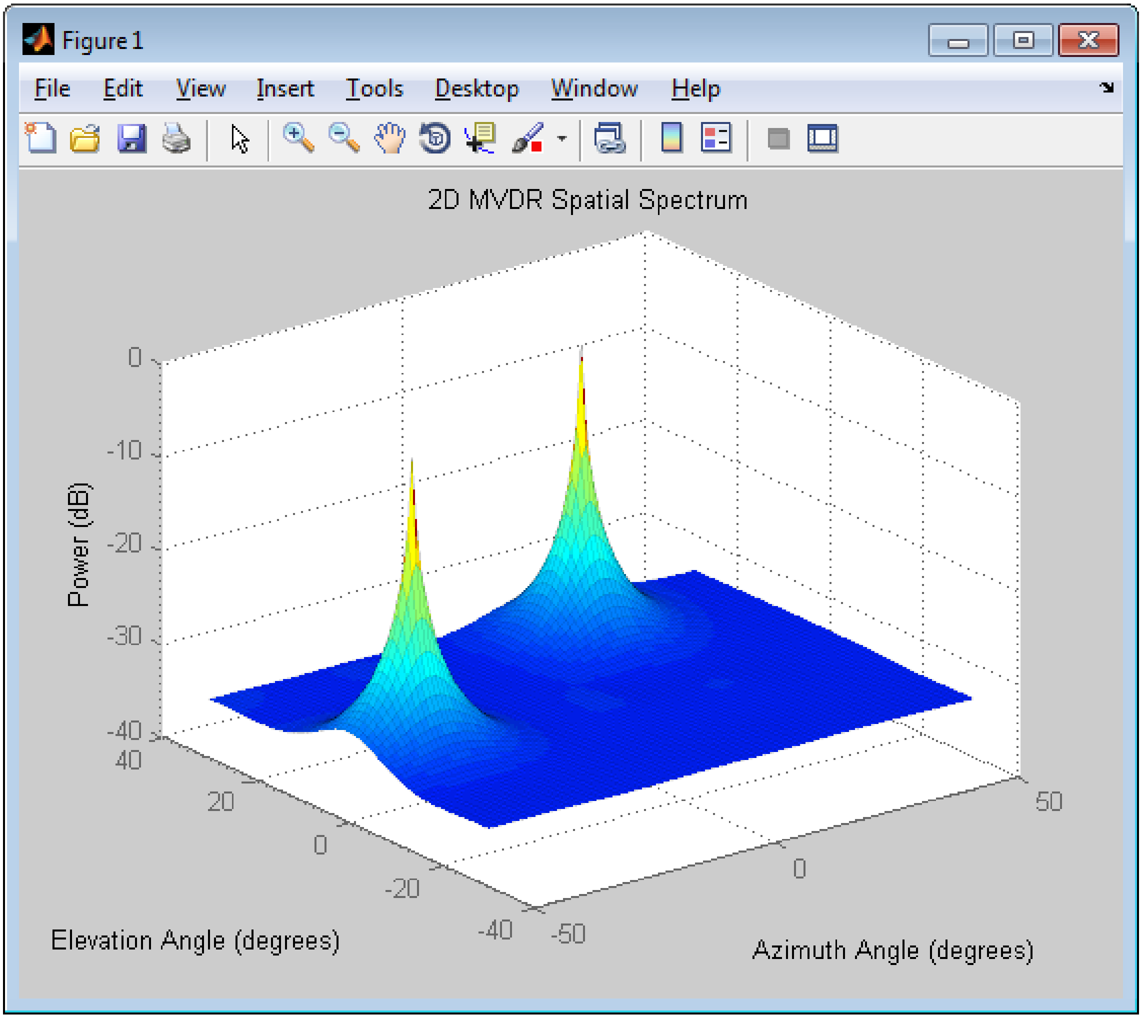


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
2D MVDR Spatial Spectrum
Power (dB)
0
-10
-20
-30
-40
40
20
0
-20
-40
Elevation Angle (degrees)
-50
0
50
Azimuth Angle (degrees)

phased.MVDREstimator2D.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.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`.

phased.MVDREstimator2D.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. 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 1-718.
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 Hz) of the microphone element as a 1x2 row vector in the form of `[LowerBound HigherBound]`. The microphone element has no response outside the specified frequency range.

Default: `[0 1e20]`

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
isPolarizationCapable	Polarization capability
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');
```

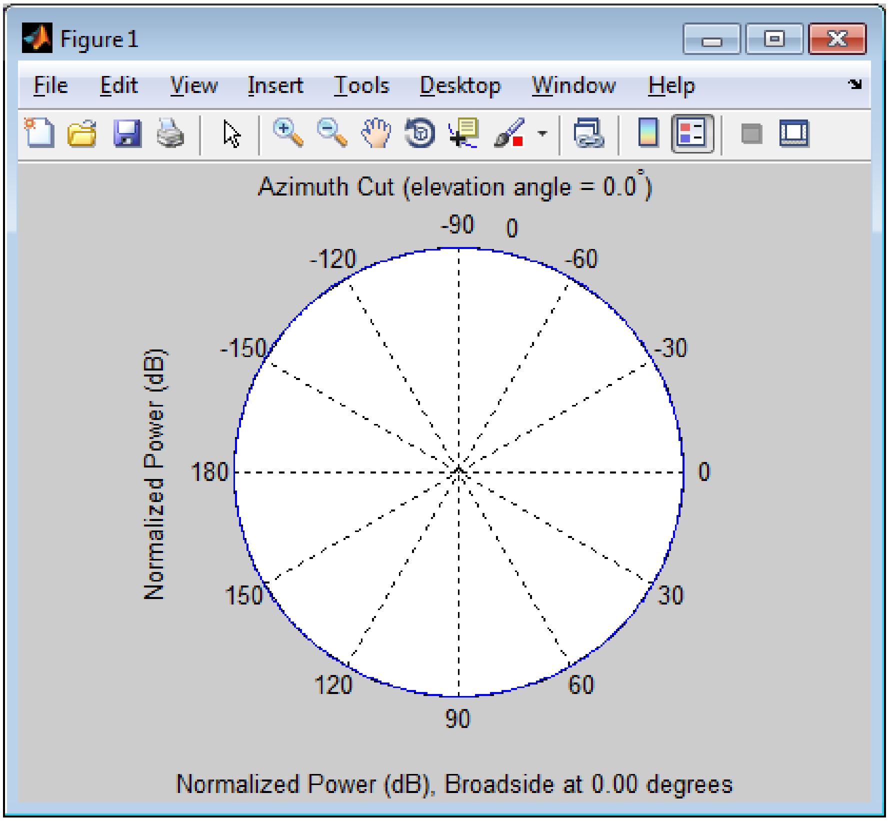


Figure 1
File Edit View Insert Tools Desktop Window Help
Azimuth Cut (elevation angle = 0.0°)
-90
0
-120
-60
-150
-30
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

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.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.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.isPolarization

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.OmnidirectionalMicrophoneElement` supports polarization. An element supports polarization if it can create or respond to polarized fields. This microphone element, as all microphone elements, does not support polarization.

Input Arguments

h - Omni-directional microphone element

Omni-directional microphone element specified as a `phased.OmnidirectionalMicrophoneElement` System object

Output Arguments

flag - Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the microphone element supports polarization or `false` if it does not. Because the `phased.OmnidirectionalMicrophoneElement` object does not support polarization, `flag` is always returned as `false`.

Examples

Omnidirectional Microphone Element does not Support Polarization

Determine whether a `phased.OmnidirectionalMicrophoneElement` microphone element supports polarization.

```
h = phased.OmnidirectionalMicrophoneElement;
isPolarizationCapable(h)
```

```
ans =

     0
```

The returned value `false (0)` shows that the omnidirectional microphone element does not support polarization.

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 System object

FREQ

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the antenna response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'AzimuthAngles'

Azimuth angles for plotting element response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting element response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting element response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting element response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Plot Response and Directivity of Omnidirectional Microphone

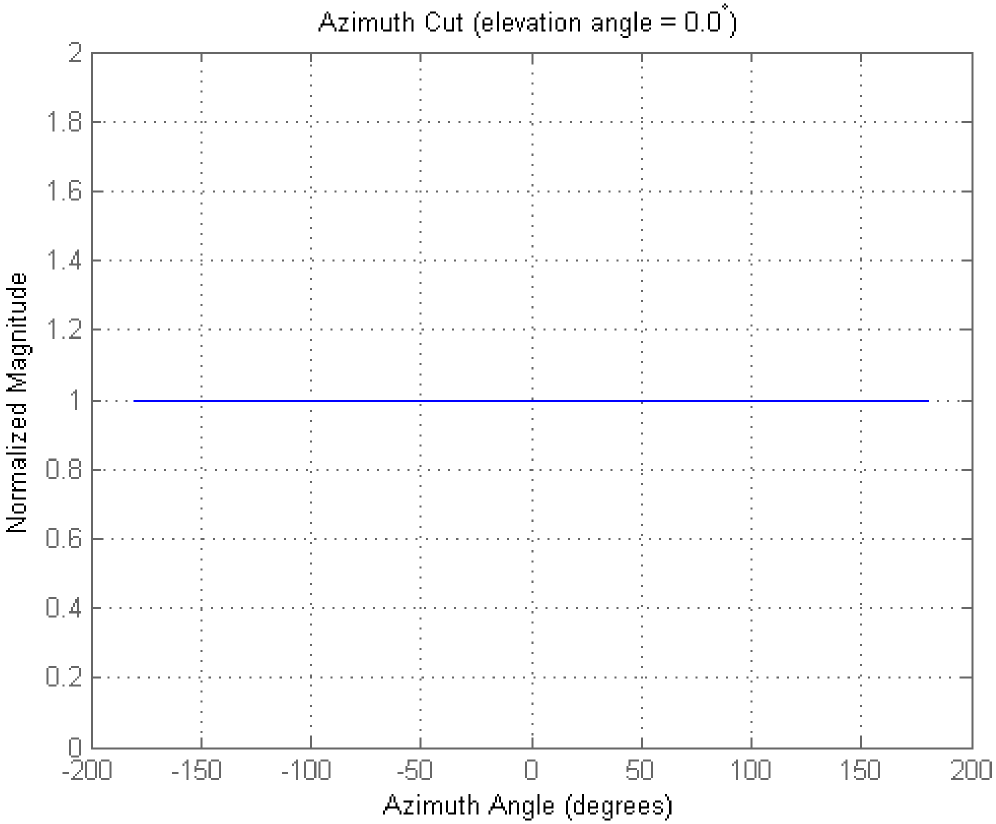
This example shows how to construct an omnidirectional microphone and how to plot its response and directivity. The microphone operating frequency spans the range 20 to 20000 Hz.

Construct the omnidirectional microphone.

```
sOmni = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
```

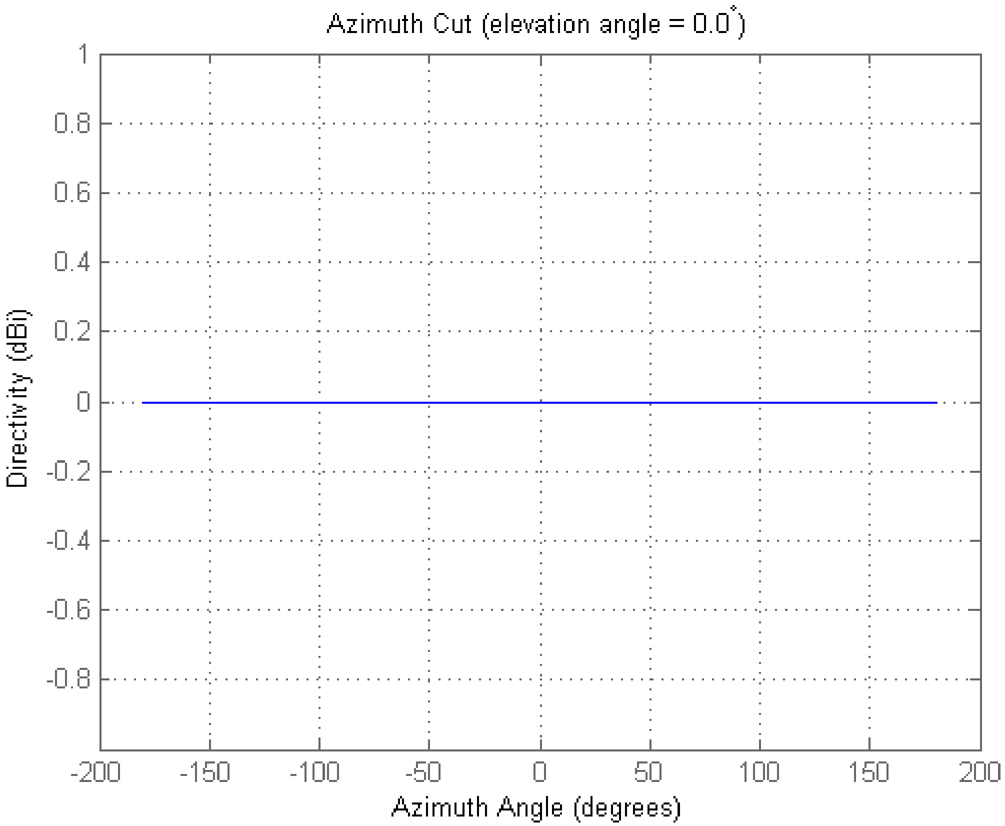
Plot the microphone response at 200 Hz.

```
fc = 200;
plotResponse(sOmni,fc,'Unit','mag');
```



Plot the microphone directivity.

```
plotResponse(sOmni,fc,'Unit','dbi');
```



Plot 3-D Response of Omnidirectional Microphone

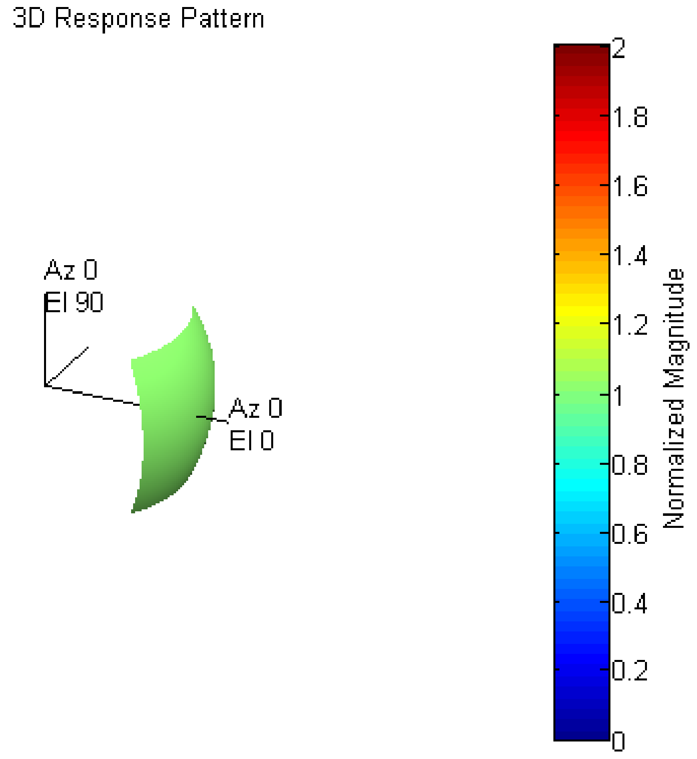
This example shows how to construct an omnidirection microphone with response in the frequency range 20 - 20000 Hz and how to plot its 3-D response over a range of angles.

Construct the microphone element.

```
sOmin = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20e3]);
```

Plot the 3-D response at 500 Hz. Show the response between -30 to 30 degrees in both azimuth and elevation in 0.1 degree increments.

```
plotResponse(sOmin,500,'Format','Polar',...
    'RespCut','3D','Unit','mag',...
    'AzimuthAngles',[-30:0.1:30],...
    'ElevationAngles',[-30:0.1:30]);
```



See Also `uv2azel` | `azel2uv`

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.OmnidirectionalMicrophoneElement.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

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');
```

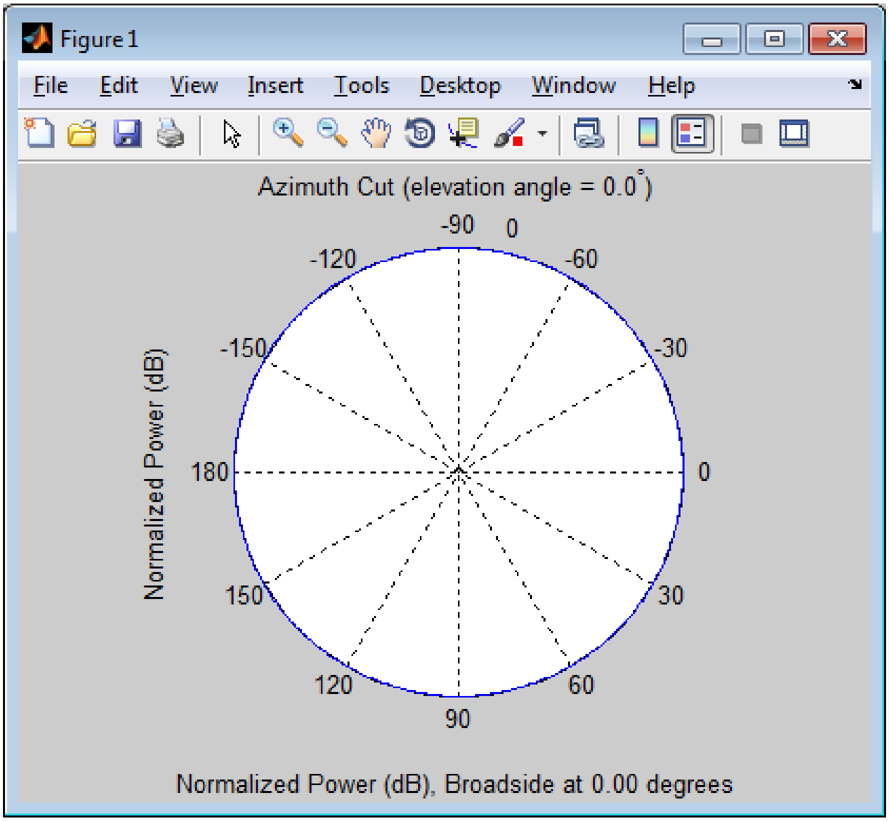


Figure 1
File Edit View Insert Tools Desktop Window Help
Azimuth Cut (elevation angle = 0.0°)
-90
0
-120
-60
-150
-30
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

See Also `uv2azel` | `phitheta2azel`

phased.PartitionedArray

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 1-739.
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
isPolarizationCapable	Polarization capability
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

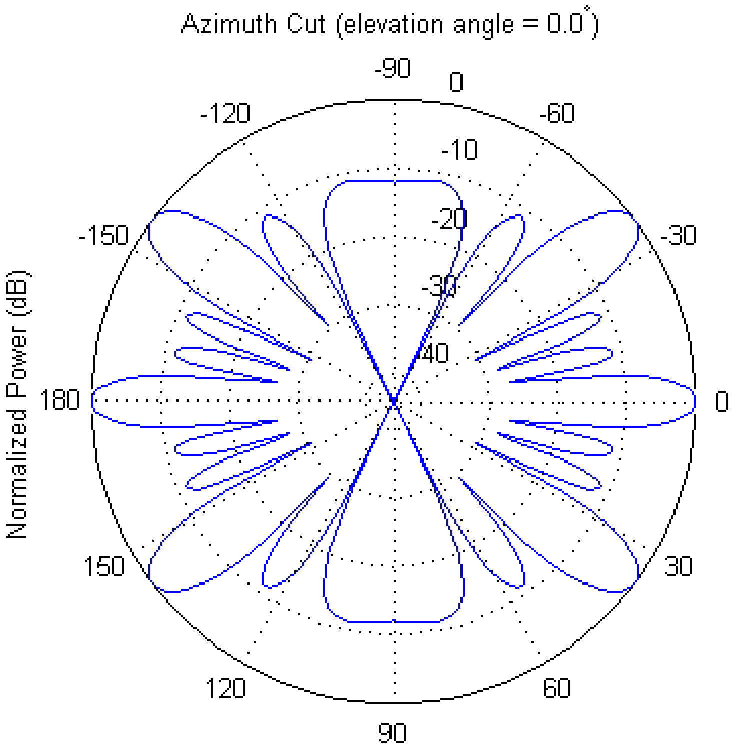
Plot the azimuth response of a 4-element ULA partitioned into two 2-element ULA’s. The element spacing is one-half wavelength.

Create the ULA, and partition it into 2-element ULA’s.

```
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');
```



Normalized Power (dB), Broadside at 0.00 degrees

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);
```

References

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

See Also

`phased.ULA` | `phased.URA` | `phased.ConformalArray` | `phased.ReplicatedSubarray`

Related Examples

- Subarrays in Phased Array Antennas
- Phased Array Gallery

Concepts

- "Subarrays Within Arrays"

phased.PartitionedArray.clone

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`.

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.

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`

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.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all its constituent sensor elements support polarization.

Input Arguments

h - Partitioned array

Partitioned array specified as a `phased.PartitionedArray` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability flag returned as a Boolean value. This value is `true`, if the array supports polarization or `false`, if it does not.

Examples

Partitioned Array of Short-Dipole Antenna Elements Supports Polarization

Determine whether a partitioned array of `phased.ShortDipoleAntennaElement` short-dipole antenna elements supports polarization.

```
hsd = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[1e9 10e9]);
ha = phased.ULA(4,'Element',hsd);
hp = phased.PartitionedArray('Array',ha,...
     'SubarraySelection',[1 1 0 0; 0 0 1 1]);
isPolarizationCapable(hp)
```

```
ans =

     1
```

The returned value `true (1)` shows that this array supports polarization.

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. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, then `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and `RespCut` is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the array response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where:

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'SteerAng'

Subarray steering angle. `SteerAng` can be either a 2-element column vector or a scalar.

If `SteerAng` is a 2-element column vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If `SteerAng` is a scalar, it specifies the azimuth angle. In this case, the elevation angle is assumed to be 0.

This option is applicable only if the `SubarraySteering` property of `H` is `'Phase'` or `'Time'`.

Default: `[0;0]`

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'Weights'

Weight values applied to the array, specified as a length-N column vector or N-by-M matrix. The dimension N is the number of subarrays in the array. The interpretation of M depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimension	**FREQ Dimension**	**Purpose**
N-by-1 column vector	Scalar or 1-by-*M* row vector	Apply one set of weights for the same single frequency or all *M* frequencies.
N-by-*M* matrix	Scalar	Apply all of the *M* different columns in `Weights` for the same single frequency.
	1-by-*M* row vector	Apply each of the *M* different columns in `Weights` for the corresponding frequency in `FREQ`.

'AzimuthAngles'

Azimuth angles for plotting subarray response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting subarray response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting subarray response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the U coordinates for visualizing the radiation pattern in U/V space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting subarray response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the V coordinates for visualizing the radiation pattern in U/V space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Azimuth Response of Partitioned ULA

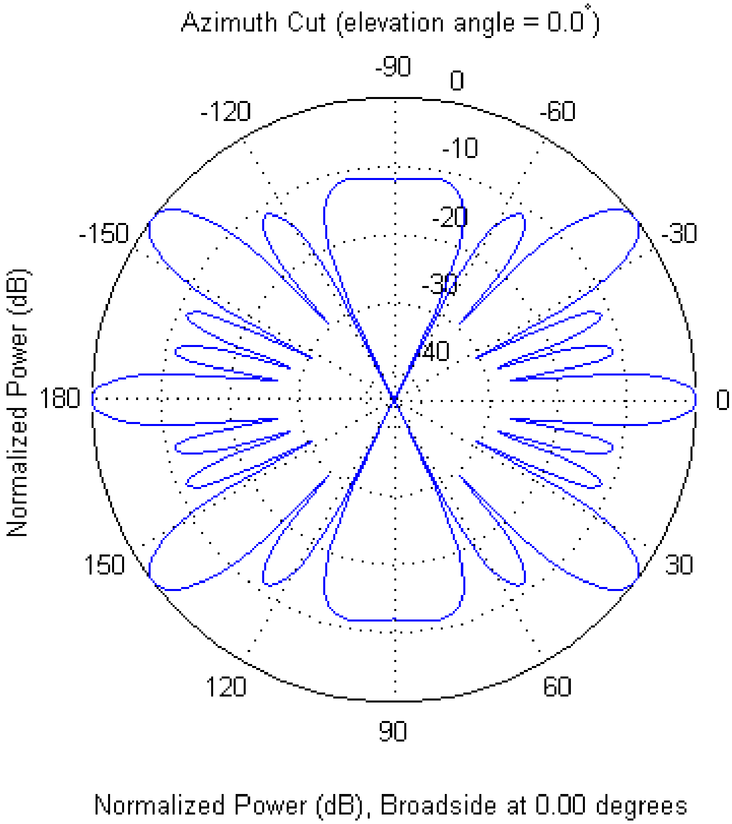
Plot the azimuth response of a 4-element ULA partitioned into two 2-element ULA's. The element spacing is one-half wavelength.

Create the ULA, and partition it into 2-element ULA's.

```
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');
```



Plot Response and Directivity of Partitioned URA Over Restricted Range of Angles

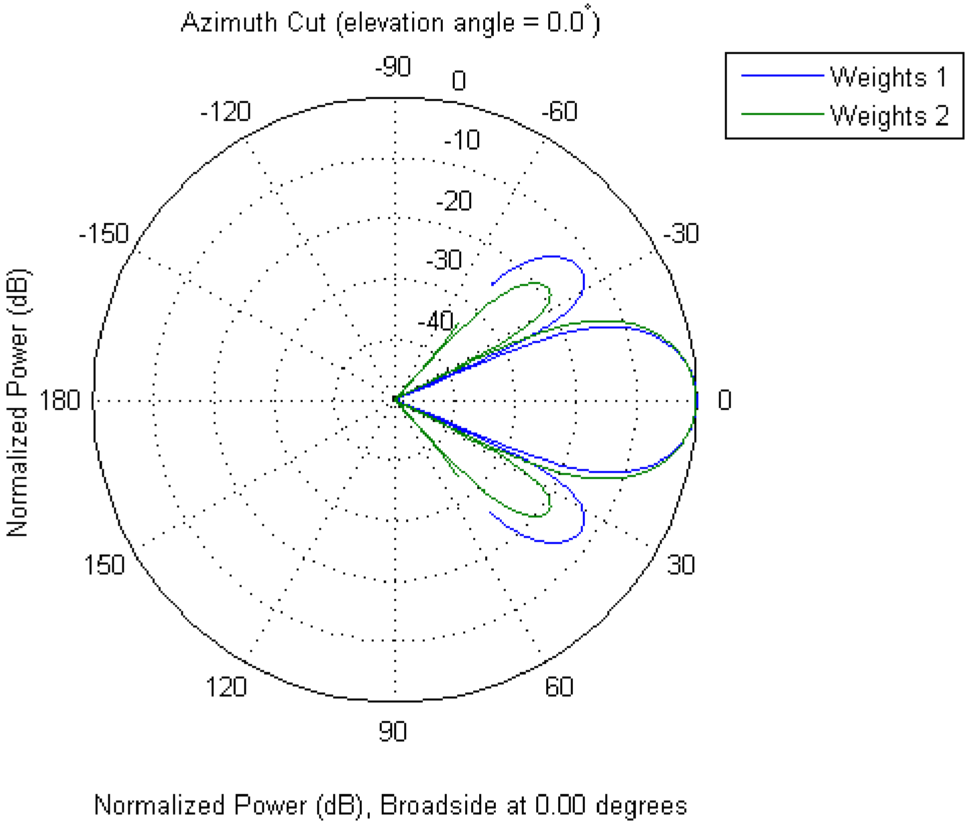
Convert a 2-by-6 URA of isotropic antenna elements into a 1-by-3 partitioned array so that each subarray of the partitioned array is a 2-by-2 URA. Assume that the frequency response of the elements lies between 1 and 6 GHz. The elements are spaced one-half wavelength apart corresponding to the highest frequency of the element response. Plot an azimuth cut from -50 to 50 degrees for different two sets of weights. For partitioned arrays, weights are applied to the subarrays instead of the elements.

Set up the partitioned array.

```
fmin = 1e9;
fmax = 6e9;
c = physconst('LightSpeed');
lam = c/fmax;
s_iso = phased.IsotropicAntennaElement(...
    'FrequencyRange',[fmin,fmax],...
    'BackBaffled',false);
s_ura = phased.URA('Element',s_iso,'Size',[2,6],...
    'ElementSpacing',[lam/2,lam/2]);
subarraymap = [[1,1,1,1,0,0,0,0,0,0,0,0];...
    [0,0,0,0,1,1,1,1,0,0,0,0];...
    [0,0,0,0,0,0,0,0,1,1,1,1]];
s_pa = phased.PartitionedArray('Array',s_ura,...
    'SubarraySelection',subarraymap);
```

Plot the response of the array at 5 GHz over the restricted range of azimuth angles.

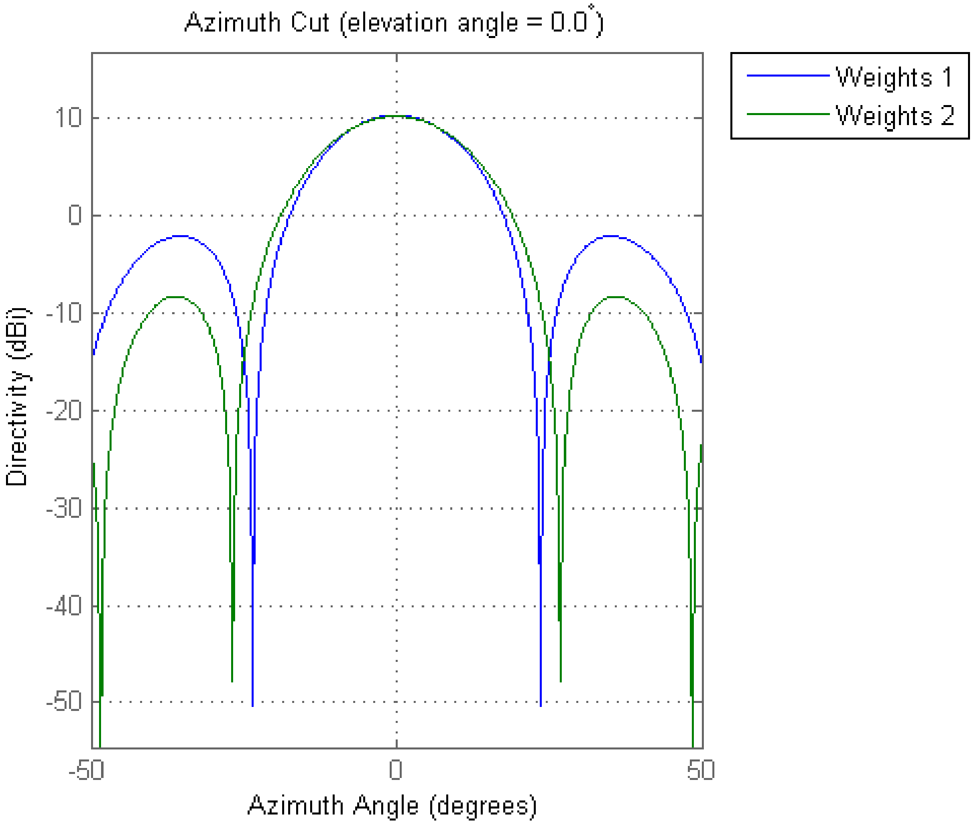
```
fc = 5e9;
wts = [[1,1,1]',[.862,1.23,.862]'];
plotResponse(s_pa,fc,c,'RespCut','Az',...
    'AzimuthAngles',[-50:0.1:50],...
    'Unit','db','Format','Polar',...
    'Weights',wts);
```



The plot of the response shows the broadening of the main lobe and the reduction of the strength of the sidelobes caused by the weight tapering.

Next, plot an azimuth cut of the directivity of the array at 5 GHz over the restricted range of azimuth angles for the two different sets of weights.

```
fc = 5e9;
wts = [[1,1,1]',[.862,1.23,.862]'];
plotResponse(s_pa,fc,c,'RespCut','Az',...
'AzimuthAngles',[-50:0.1:50],...
'Unit','dbi',...
'Weights',wts);
```



See Also `uv2azel` | `azel2uv`

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.

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

Voltage responses of the subarrays of a phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions N-by-M-by-L. The size N represents the number of subarrays in the phased array, M represents the number of angles specified in `ANG`, and L represents the number of frequencies specified in `FREQ`.

For a particular subarray, each column of `RESP` contains the responses of the subarray for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP` contains the responses of the subarrays for the corresponding frequency specified in `FREQ`.

- If the array is capable of supporting polarization, the voltage response, `RESP`, is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field `RESP.H` represents the array's horizontal polarization response while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*. The size *N* represents the number of subarrays in the phased array, *M* represents the number of angles specified in `ANG`, and *L* represents the number of frequencies specified in `FREQ`. For a particular subarray, each column of `RESP` contains the responses of the subarray for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP` contains the responses of the subarrays for the corresponding frequency specified in `FREQ`.

Examples

Response of Subarrays in Partitioned ULA

Calculate the response at 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`

phased.PartitionedArray.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. `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`

'ShowTaper'

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color. The default value is `false`.

Default: `false`

'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

Highlight Overlapped Subarrays

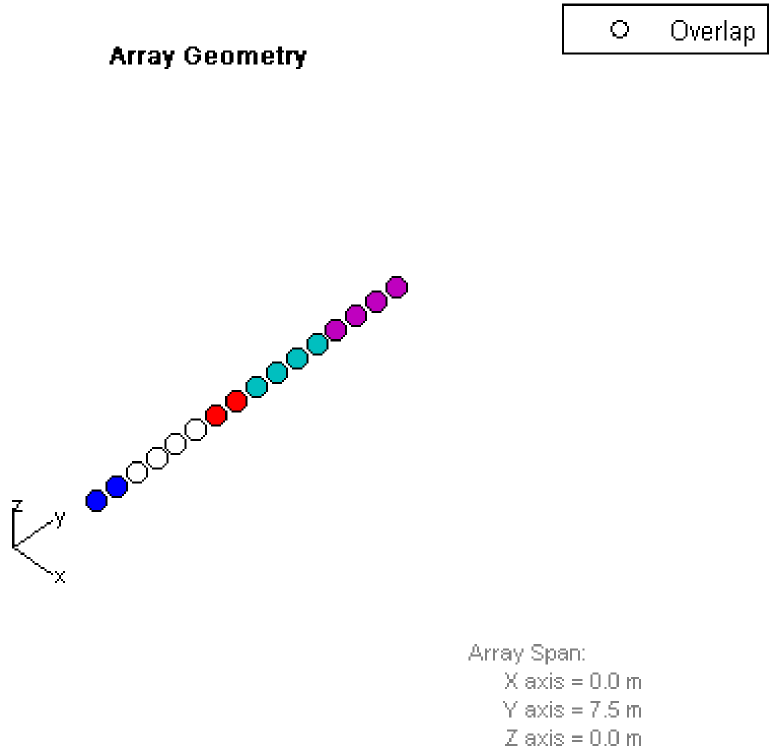
Display the geometry of a uniform linear array having overlapped subarrays.

Create a 16-element ULA that has five 4-element subarrays. Some elements occur in more than one subarray.

```
h = phased.ULA(16);
ha = phased.PartitionedArray('Array',h,...
    'SubarraySelection',...
    [1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0;...
    0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0;...
    0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0;...
    0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0;...
    0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1]);
```

Display the geometry of the array, highlighting all subarrays.

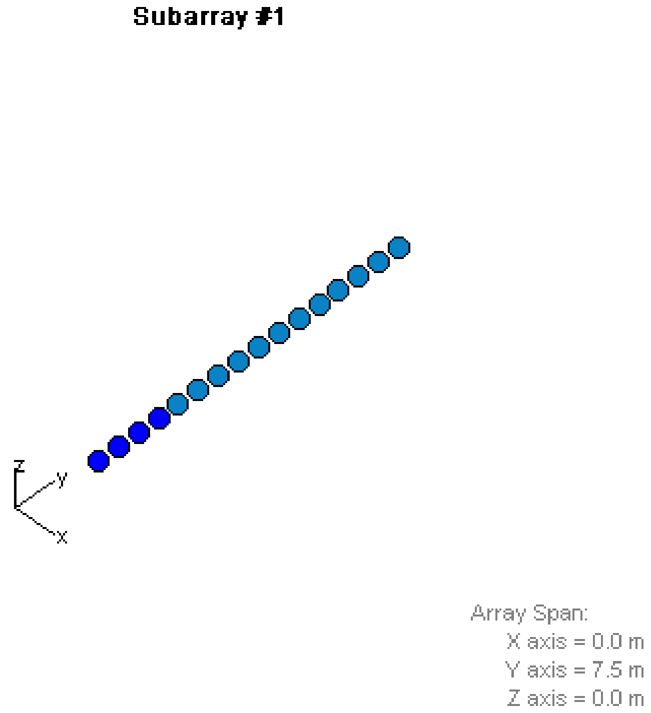
```
viewArray(ha);
```

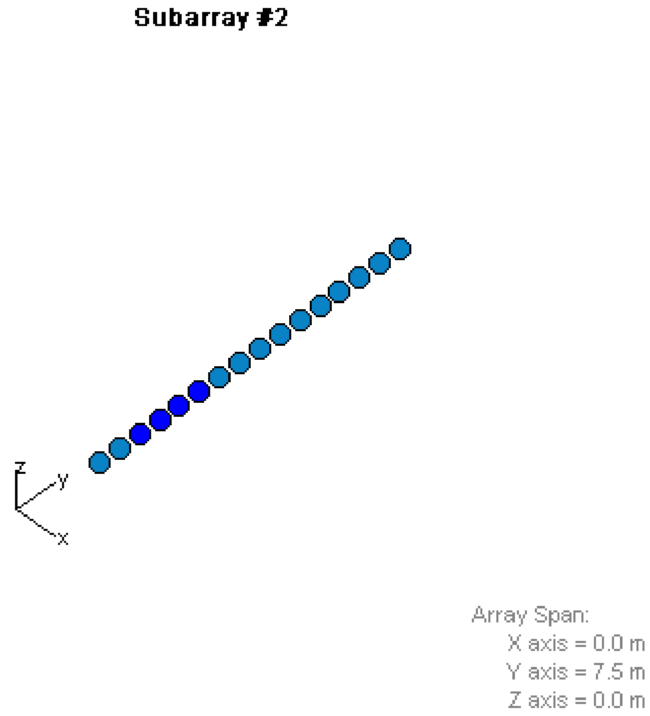

Array Geometry
○ Overlap
z
y
x
Array Span:
X axis = 0.0 m
Y axis = 7.5 m
Z axis = 0.0 m

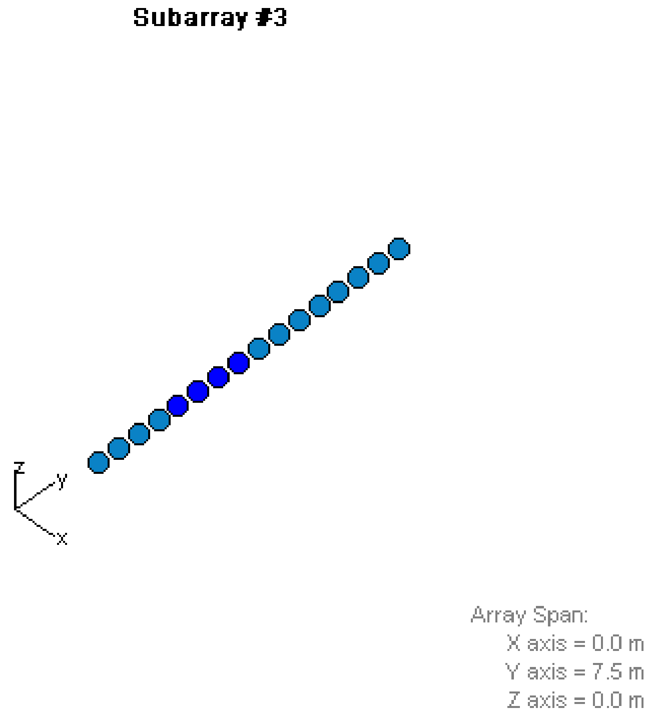
Each color other than white represents a different subarray. White represents elements that occur in multiple subarrays.

Examine the overlapped subarrays by creating separate figures that highlight the first, second, and third subarrays. In each figure, dark blue represents the highlighted elements.

```
for idx = 1:3
    figure;
    viewArray(ha,'ShowSubarray',idx,...
            'Title',['Subarray #' num2str(idx)]);
end
```




Subarray #2
z
y
x
Array Span:
X axis = 0.0 m
Y axis = 7.5 m
Z axis = 0.0 m



See Also `phased.ArrayResponse`

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 1-779.
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`

Code

Phase code type

Specify the phase code type used in phase modulation. Valid values are:

- `'Barker'`
- `'Frank'`
- `'P1'`
- `'P2'`
- `'P3'`
- `'P4'`
- `'Px'`
- `'Zadoff-Chu'`

Default: `'Frank'`

ChipWidth

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 the `Code` property is ...	Then the `NumChips` property must be...
`'Frank'`, `'P1'`, or `'Px'`	A perfect square
`'P2'`	An even number that is a perfect square
`'Barker'`	2, 3, 4, 5, 7, 11, or 13

Default: `4`

SequenceIndex

Zadoff-Chu sequence index

Specify the sequence index used in Zadoff-Chu code as a positive integer. This property applies only when you set the `Code` property to `'Zadoff-Chu'`. The value of `SequenceIndex` must be relatively prime to the value of the `NumChips` property.

Default: `1`

PRF

Pulse repetition frequency

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('Code','Zadoff-Chu',...
    'ChipWidth',1e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
plot(hw);
```

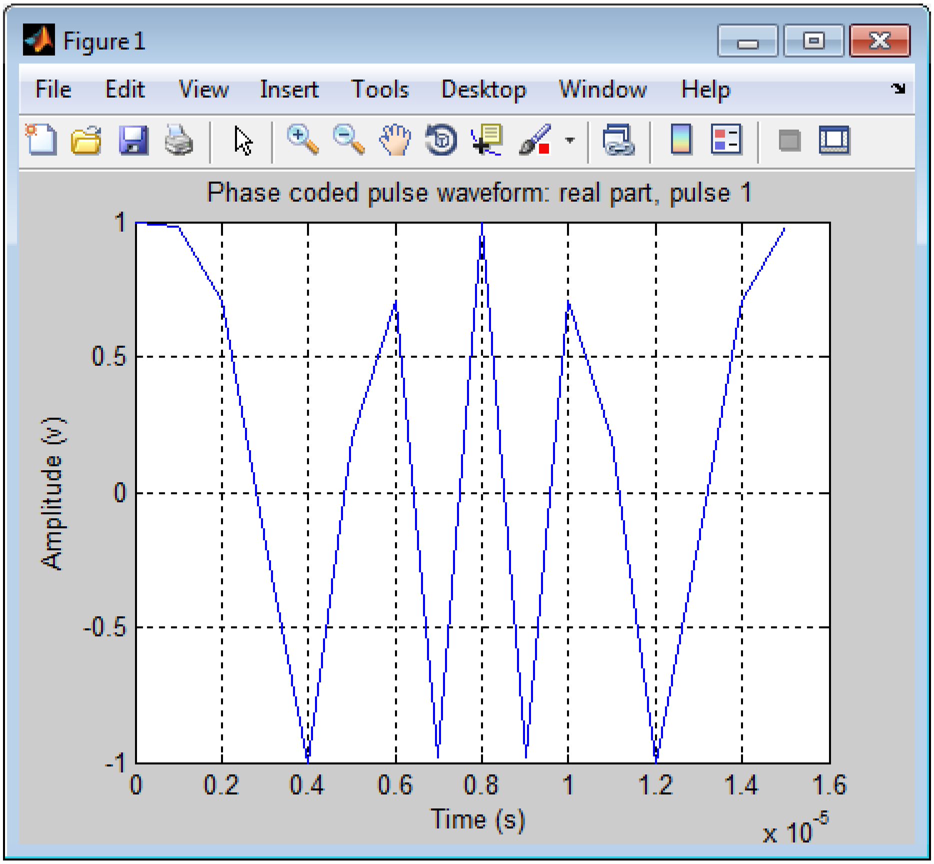


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Phase coded pulse waveform: real part, pulse 1
Amplitude (v)
1
0.5
0
-0.5
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
Time (s)
x 10^-5

Generate samples of a phase-coded pulse waveform that uses the Zadoff-Chu code, and plot the samples.

```
hw = phased.PhaseCodedWaveform('Code','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)');
```

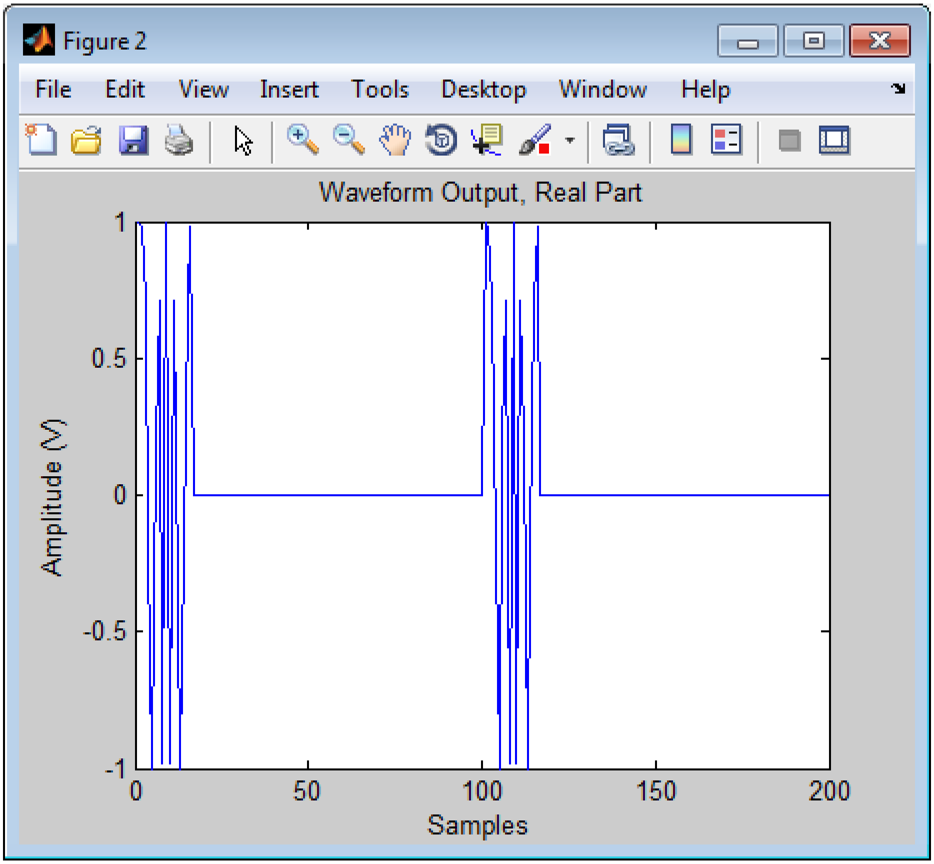


Figure 2
File Edit View Insert Tools Desktop Window Help
Waveform Output, Real Part
Amplitude (V)
1
0.5
0
-0.5
-1
0
50
100
150
200
Samples

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('Code','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])
```

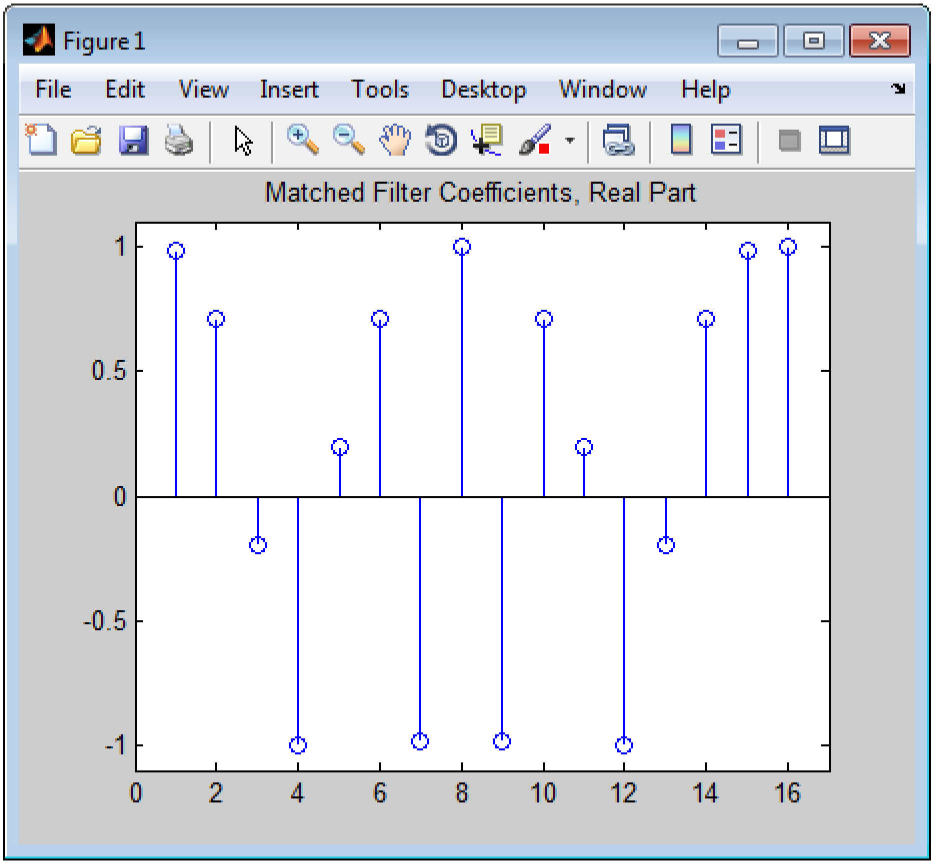


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Matched Filter Coefficients, Real Part
1
0.5
0
-0.5
-1
0
2
4
6
8
10
12
14
16

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.

phased.PhaseCodedWaveform.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 `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 `PlotType` 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. `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('Code','Zadoff-Chu',...
    'ChipWidth',1e-6,'NumChips',16,...
    'OutputFormat','Pulses','NumPulses',2);
plot(hw);
```

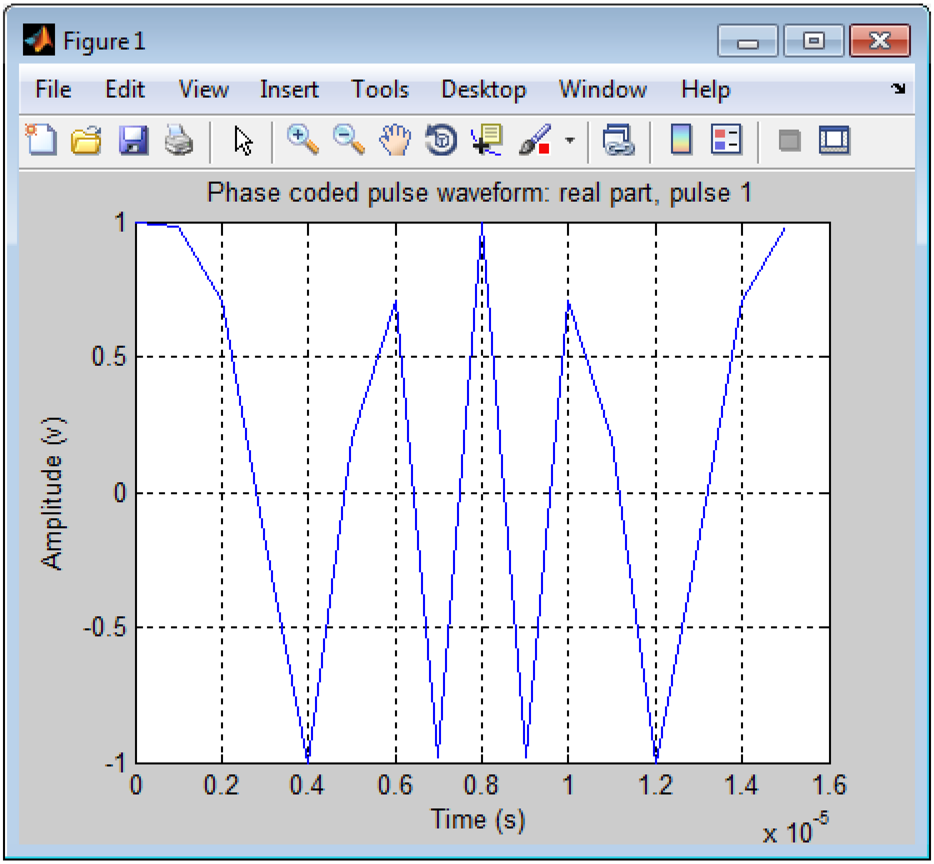


Figure 1
File Edit View Insert Tools Desktop Window Help
Phase coded pulse waveform: real part, pulse 1
1
0.5
0
-0.5
-1
Amplitude (v)
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
Time (s)
x 10^-5

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('Code','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)');
```

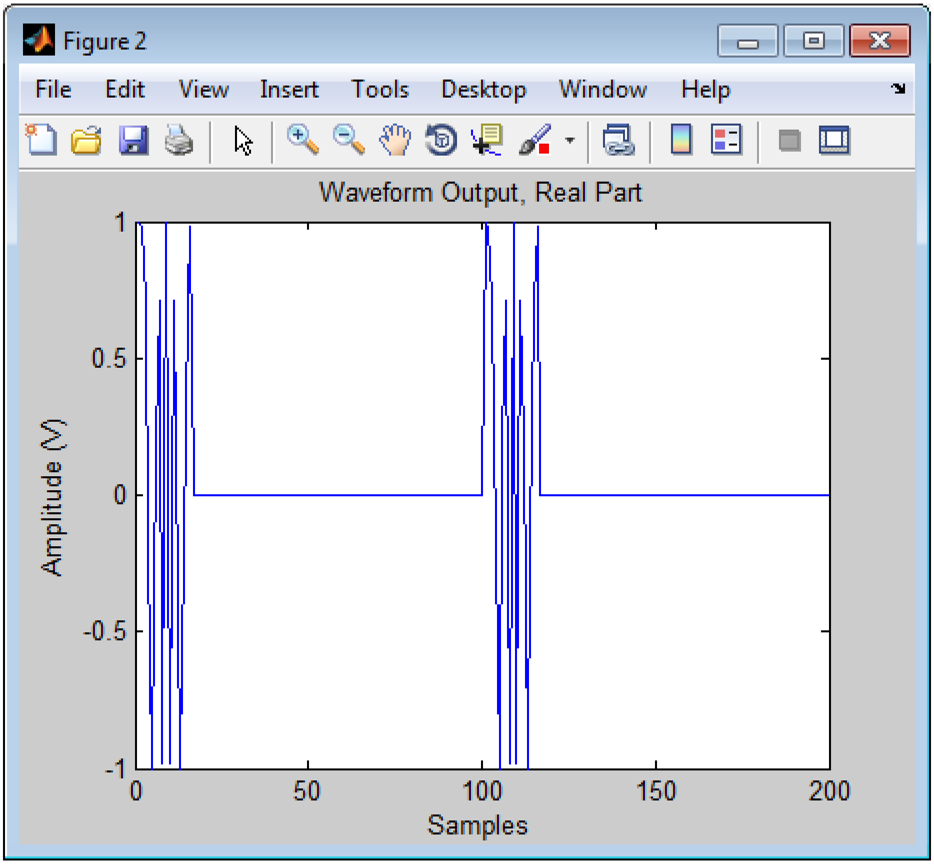


Figure 2
File
Edit
View
Insert
Tools
Desktop
Window
Help
Waveform Output, Real Part
Amplitude (V)
Samples
1
0.5
0
-0.5
-1
0
50
100
150
200

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 1-801.
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

Sensor array

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

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

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);
```

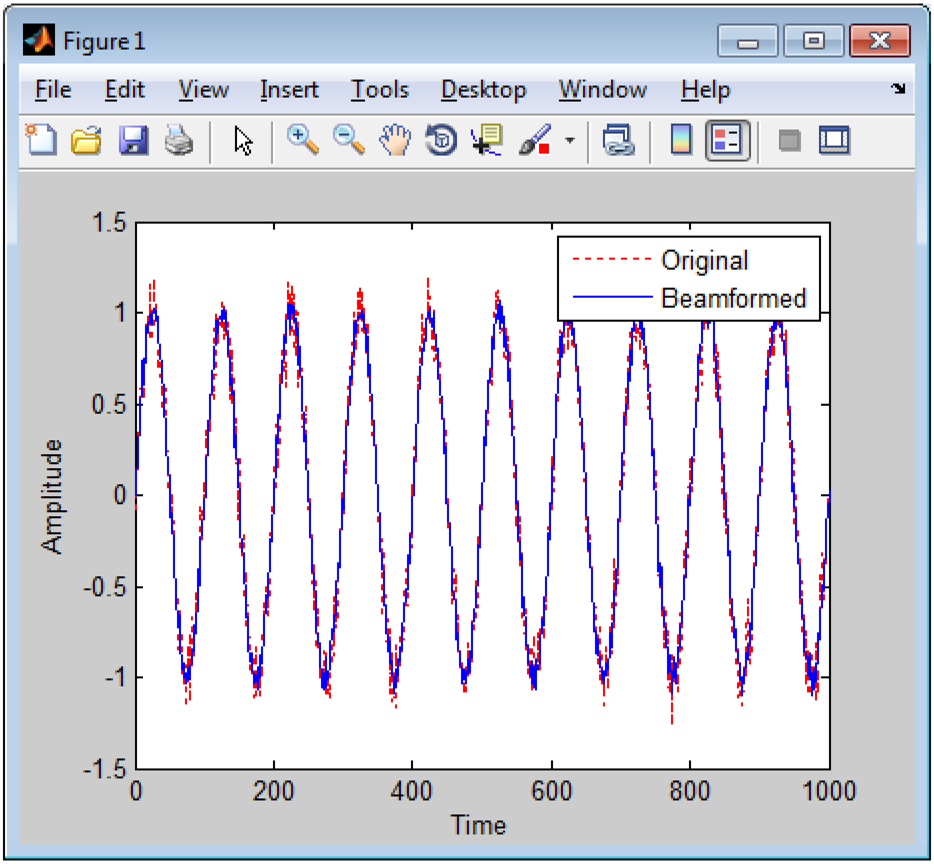
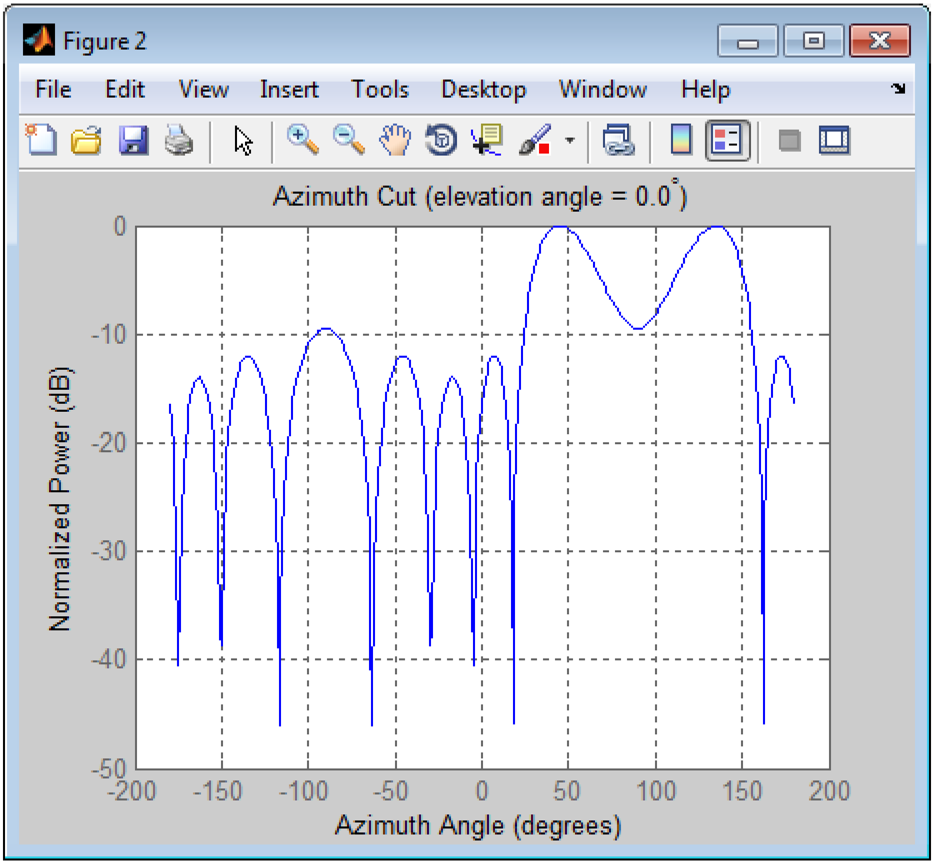


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Original
Beamformed
Amplitude
Time
1.5
1
0.5
0
-0.5
-1
-1.5
0
200
400
600
800
1000



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.

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 1-816.
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` **|** `local2globalcoord` **|** `phased.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.

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)
```

phased.RadarTarget

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 1-826.
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

EnablePolarization

Allow polarized signals

Set this property to `true` to allow the target to simulate the reflection of polarized radiation. Set this property to `false` to ignore polarization.

Default: `false`

Mode

Target scattering mode

Target scattering mode specified as one of `'Monostatic'` or `'Bistatic'`. If you set this property to `'Monostatic'`, the signal's reflection direction is the opposite to its incoming direction. If you set this property to `'Bistatic'`, the signal's reflection direction differs from its incoming direction. This property applies when you set the `EnablePolarization` property to `true`.

Default: `'Monostatic'`

ScatteringMatrixSource

Source of target mean scattering matrix

Source of target mean scattering matrix specified as one of `'Property'` or `'Input port'`. If you set the `ScatteringMatrixSource` property to `'Property'`, the target's mean scattering matrix is determined by the value of the `ScatteringMatrix` property. If you set this property to `'Input port'`, the mean scattering matrix is determined by an input argument of the `step` method. This property applies only when you set the `EnablePolarization` property to `true`. When the `EnablePolarization` property is set to `false`, use the `MeanRCSSource` property instead, together with the `MeanRCS` property, if needed.

Default: `'Property'`

ScatteringMatrix

Mean radar scattering matrix

Mean radar scattering matrix specified as a 2-by-2 matrix. This matrix represents the mean value of the target's radar cross-section (in square meters). The matrix has the form `[s_hh s_hv;s_vh s_vv]`. In this matrix, the component `s_hv` specifies the complex scattering response when the input signal is vertically polarized and the reflected signal is horizontally polarized. The other components are defined similarly. This property applies when you set the `ScatteringMatrixSource` property to `'Property'` and the `EnablePolarization` property to `true`. When the `EnablePolarization` property is set to `false`, use the `MeanRCS` property instead, together with the `MeanRCSSource` property. This property is tunable.

Default: `[1 0;0 1]`

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.

When `EnablePolarization` property is set to `true`, use the `ScatteringMatrixSource` property together with the `ScatteringMatrix` property if needed.

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.

When `EnablePolarization` property is set to `true`, use the `ScatteringMatrix` property together with the `ScatteringMatrixSource`.

Default: `1`

Model

Target statistical model

Specify the statistical model of the target as one of `'Nonfluctuating'`, `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If you set this property to a value other than `'Nonfluctuating'`, you must use the `UPDATERCS` input argument when invoking `step`.

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

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

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

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

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

References

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

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

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

See Also

`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)

Y = step(H,X,ANGLE_IN,LAXES)
Y = step(H,X,ANGLE_IN,ANGLE_OUT,LAXES)
Y = step(H,X,ANGLE_IN,LAXES,SMAT)
Y = step(H,X,ANGLE_IN,LAXES,UPDATESMAT)
Y = step(H,X,ANGLE_IN,ANGLE_OUT,LAXES,SMAT,UPDATESMAT)
```

Description

`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 *Radar cross-section* (RCS) value. This syntax applies only when the `EnablePolarization` property is set to `false`.

`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. This syntax applies only when the `EnablePolarization` property is set to `false`.

`Y = step(H,X,UPDATERCS)` uses `UPDATERCS` as the indicator of whether to update the RCS value. This syntax is available when you set the `Model` property to `'Swerling1'`, `'Swerling2'`, `'Swerling3'`, or `'Swerling4'`. If `UPDATERCS` is `true`, a new RCS value is generated. If `UPDATERCS` is `false`, the previous RCS value is used. This syntax applies only when the `EnablePolarization` property is set to `false`.

`Y = step(H,X,MEANRCS,UPDATERCS)` lets you can combine optional input arguments when their enabling properties are set. This syntax applies only when the `EnablePolarization` property is set to `false`.

`Y = step(H,X,ANGLE_IN,LAXES)` returns the reflected signal `Y` from an incident signal `X`. This syntax applies only when the `EnablePolarization` property is set to `true`. The input argument, `ANGLE_IN`, specifies the direction of the incident signal with respect to the target's local coordinate system. The input argument, `LAXES`, specifies the direction of the local coordinate axes with respect to the global coordinate system. This syntax requires that you set the `Model` property to `'Nonfluctuating'` and the `Mode` property to `'Monostatic'`. In this case, the value of the `ScatteringMatrix` property is used as the scattering matrix value.

`X` is a row array of MATLAB `struct` type, each member of the array representing a different signal. The `struct` contains three fields, `X.X`, `X.Y`, and `X.Z`. Each field corresponds to the *x*, *y*, and *z* components of the polarized input signal. Polarization components are measured with respect to the global coordinate system. Each field is a column vector representing a sequence of values for each incoming signal. The `X.X`, `X.Y`, and `Y.Z` fields must all have the same dimension. The argument, `ANGLE_IN`, is a 2-row matrix representing the signals' incoming directions with respect to the target's local coordinate system. Each column of `ANGLE_IN` specifies the incident direction of the corresponding signal in the form `[AzimuthAngle; ElevationAngle]`. Angle units are in degrees. The number of columns in `ANGLE_IN` must equal the number of members in the `X` array. The argument, `LAXES,` is a 3-by-3 matrix. Each column is a unit vector specifying the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively, with respect to the global coordinate system. Each columns is written in `[x;y;z]` form.

`Y` is a row array of `struct` type having the same size as `X`. Each `struct` contains the three reflected polarized fields, `Y.X`, `Y.Y`, and `Y.Z`. Each field corresponds to the *x*, *y*, and *z* component of the signal. Polarization components are measured with respect to the global coordinate system. Each field is a column vector representing one reflected signal.

`Y = step(H,X,ANGLE_IN,ANGLE_OUT,LAXES)`, in addition, specifies the reflection angle, `ANGLE_OUT`, of the reflected signal when you set the `Mode` property to `'Bistatic'`. This syntax applies only when the

`EnablePolarization` property is set to `true`. `ANGLE_OUT` is a 2-row matrix representing the reflected direction of each signal. Each column of `ANGLE_OUT` specifies the reflected direction of the signal in the form `[AzimuthAngle; ElevationAngle]`. Angle units are in degrees. The number of columns in `ANGLE_OUT` must equal the number of members in the `X` array. The number of columns in `ANGLE_OUT` must equal the number of elements in the `X` array.

`Y = step(H,X,ANGLE_IN,LAXES,SMAT)` specifies `SMAT` as the scattering matrix. This syntax applies only when the `EnablePolarization` property is set to `true`. The input argument `SMAT` is a 2-by-2 matrix. You must set the `ScatteringMatrixSource` property `'Input port'` to use `SMAT`.

`Y = step(H,X,ANGLE_IN,LAXES,UPDATESMAT)` specifies `UPDATESMAT` to indicate whether to update the scattering matrix when you set the `Model` property to `'Swerling1'`, `'Swerling2'`', `'Swerling3'`, or `'Swerling4'`. This syntax applies only when the `EnablePolarization` property is set to `true`. If `UPDATESMAT` is set to `true`, a scattering matrix value is generated. If `UPDATESMAT` is `false`, the previous scattering matrix value is used.

You can combine optional input arguments when their enabling properties are set. Optional inputs must be listed in the same order as the order of their enabling properties. For example,`Y = step(H,X,ANGLE_IN,ANGLE_OUT,LAXES,SMAT,UPDATESMAT)`

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 m^2. The carrier frequency of the sine wave is 1 GHz.

Reflection of Sine Wave

```
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

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

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

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

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

References

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

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

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

phased.Radiator

Purpose

Narrowband signal radiator

Description

The `phased.Radiator` object implements a narrowband signal radiator. For any antenna element, microphone element, or array, the radiator creates the outgoing signal that is to be propagated to the far field using the `phased.FreeSpace` object. You can think of the output of `phased.Radiator` as the field at a reference distance from the element or center of the array. The signal can represent a polarized or nonpolarized field depending upon whether the element or array supports polarization and the value of the `EnablePolarization` property. For arrays, you can create a superposed field of all array elements signals or a separate field for each element depending upon the value of the `CombineRadiatedSignals` property.

To compute the radiated signal from the sensor(s):

1. Define and set up your radiator. See "Construction" on page 1-844.
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

Sensor element or sensor array

Sensor element or sensor array specified as a System object in the Phased Array System Toolbox. A sensor array can contain subarrays.

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

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`

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`

EnablePolarization

Enable Polarization

Set this property to `true` to simulate the radiation of polarized waves. Set this property to `false` to ignore polarization. This property applies when the sensor specified in the `Sensor` property is capable of simulating polarization.

Default: `false`

WeightsInputPort

Enable weights input

To specify weights, set this property to `true` and then 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 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 the signal from a single isotropic 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` | `phased.FreeSpace`

phased.Radiator.clone

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.

phased.Radiator.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.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.

Purpose

Radiate signals

Syntax

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

Description

`Y = step(H,X,ANG)` radiates signal `X` in the direction `ANG`. `Y` is the radiated signal. The radiating process depends on the `CombineRadiatedSignals` property of `H`, as follows:

- If `CombineRadiatedSignals` has the value `true`, each radiating element or subarray radiates `X` in all the directions in `ANG`. `Y` combines the outputs of all radiating elements or subarrays. If the `Sensor` property of `H` contains subarrays, the radiating process distributes the power equally among the elements of each subarray.
- If `CombineRadiatedSignals` has the value `false`, each radiating element radiates `X` in only one direction in `ANG`. Each column of `Y` contains the output of the corresponding element. The `false` option is available when the `Sensor` property of `H` does not contain subarrays.

`Y = step(H,X,ANG,LAXES)` uses `LAXES` as the local coordinate system axes directions. This syntax is available when you set the `EnablePolarization` property to `true`.

`Y = step(H,X,ANG,WEIGHTS)` uses `WEIGHTS` as the weight vector. This syntax is available when you set the `WeightsInputPort` property to `true`.

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

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

Note The object performs an initialization the first time the `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

Radiating directions of signals. `ANG` is a two-row matrix. Each column specifies a radiating direction in the form [AzimuthAngle; ElevationAngle], in degrees.

LAXES

Local coordinate system. `LAXES` is a 3-by-3 matrix whose columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively. Each axis is specified in terms of `[x;y;z]` with respect to the global coordinate system. This argument is only used when the `EnablePolarization` property is set to `true`.

WEIGHTS

Vector of weights. `WEIGHTS` is a column vector whose length equals the number of radiating elements or subarrays.

STEERANGLE

Subarray steering angle, specified as a length-2 column vector. The vector has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90° and 90°, inclusive.

Output Arguments

Y

Radiated signals

- If the `EnablePolarization` property value is set to `false`, the output argument `Y` is a matrix. The number of columns of the matrix equals the number of radiating signals. Each column of `Y` contains a separate radiating signal. The number of radiating signals depends upon the `CombineRadiatedSignals` property of `H`.
- If the `EnablePolarization` property value is set to `true`, `Y` is a row vector of elements of MATLAB `struct` type. The length of the `struct` vector equals the number of radiating signals. Each `struct` contains a separate radiating signal. The number of radiating signals depends upon the `CombineRadiatedSignals` property of `H`. Each `struct` contains three column-vector fields, `X`, `Y`, and `Z`. These fields represent the *x*, *y*, and *z* components of the polarized wave vector signal in the global coordinate system.

Examples

Radiating from a 5-Element ULA

Combine the radiation from five isotropic antenna elements.

Set up a uniform line array of five isotropic antennas. Then, construct the radiator object.

```
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);
```

Radiating from a 5-Element ULA of Polarized Antennas

Combine the radiation from five short-dipole antenna elements.

Set up a uniform line array of five short-dipole antennas with polarization enabled. Then, construct the radiator object.

```
hsd = phased.ShortDipoleAntennaElement;
ha = phased.ULA('Element',hsd,'NumElements',5);
hr = phased.Radiator('Sensor',ha,...
    'OperatingFrequency',300e6,'CombineRadiatedSignals',true,'EnablePolar
```

Rotate the local coordinate system by 10° around the x-axis. Demonstrate that the output represents a polarized field.

```
x = [1;1];
radiatingAngle = [30 30; 0 20];
y = step(hr,x,radiatingAngle,rotx(10))
```

```
y =

1x2 struct array with fields:
```

X
Y
Z

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 1-858.

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.

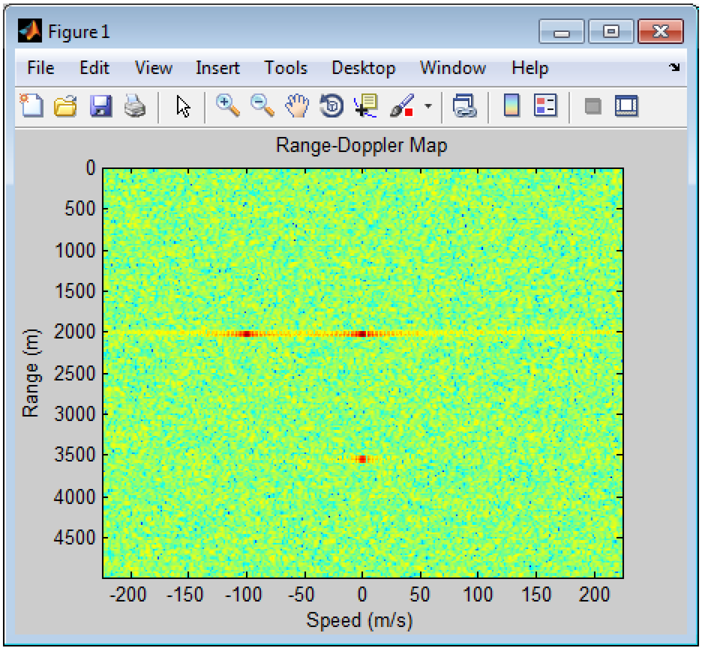
```
hrdresp = phased.RangeDopplerResponse(...
   'DopplerFFTLengthSource','Property',...
   'DopplerFFTLength',RangeDopplerEx_MF_NFFTDOP,...
   'SampleRate',RangeDopplerEx_MF_Fs,...
   'DopplerOutput','Speed',...
   'OperatingFrequency',RangeDopplerEx_MF_Fc);
```

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)
```

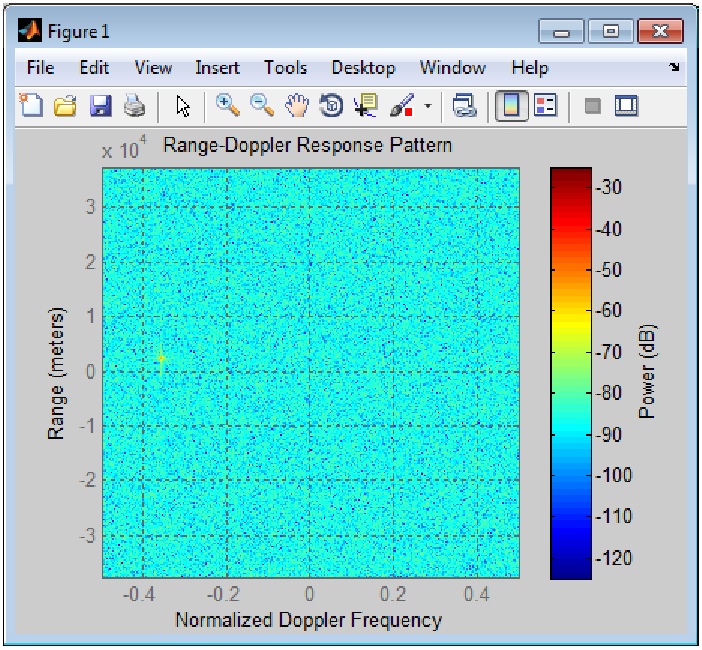


Figure 1
File Edit View Insert Tools Desktop Window Help
x 10^4
Range-Doppler Response Pattern
Range (meters)
3
2
1
0
-1
-2
-3
-0.4
-0.2
0
0.2
0.4
Normalized Doppler Frequency
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Power (dB)

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

phased.RangeDopplerResponse.clone

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`.

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. `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)
```

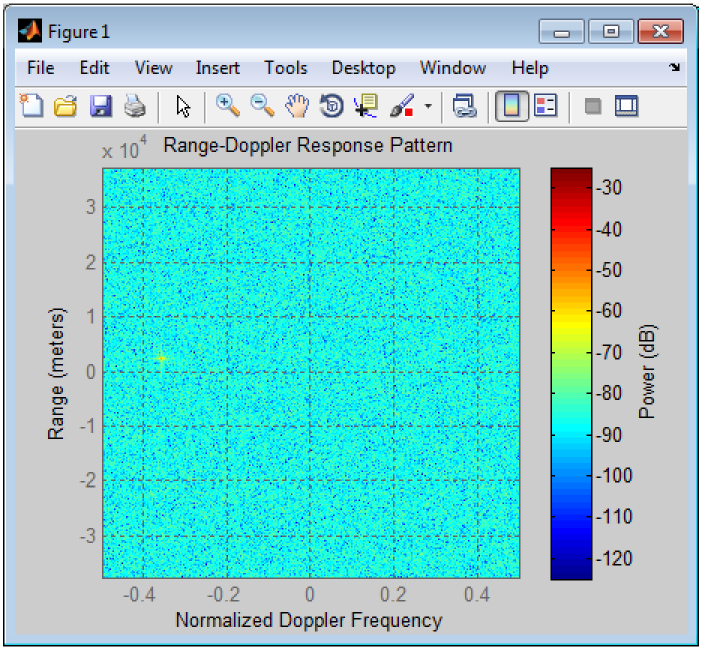


Figure 1
File Edit View Insert Tools Desktop Window Help
x 10^4
Range-Doppler Response Pattern
Range (meters)
3
2
1
0
-1
-2
-3
-0.4
-0.2
0
0.2
0.4
Normalized Doppler Frequency
-30
-40
-50
-60
-70
-80
-90
-100
-110
-120
Power (dB)

See Also `phased.AngleDopplerResponse.plotResponse`

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

phased.RangeDopplerResponse.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 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');
```

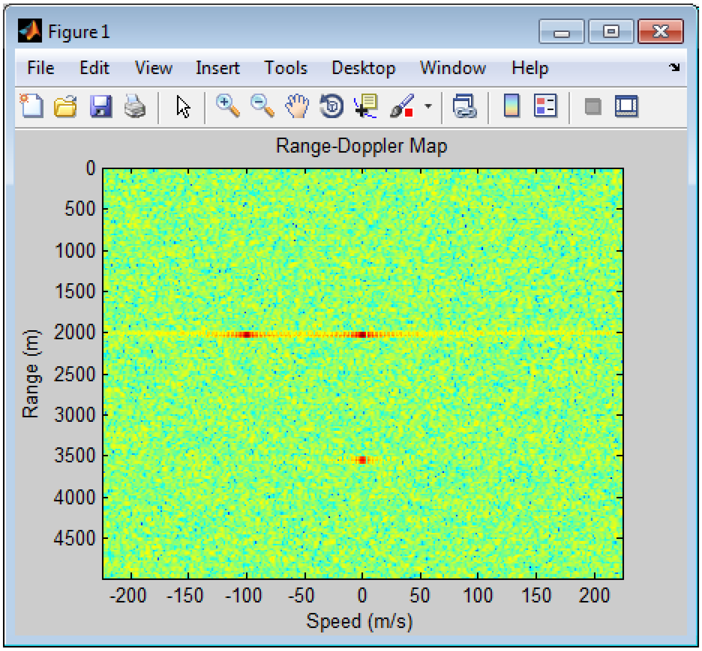


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Range-Doppler Map
0
500
1000
1500
2000
2500
3000
3500
4000
4500
Range (m)
-200
-150
-100
-50
0
50
100
150
200
Speed (m/s)

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 1-885.
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

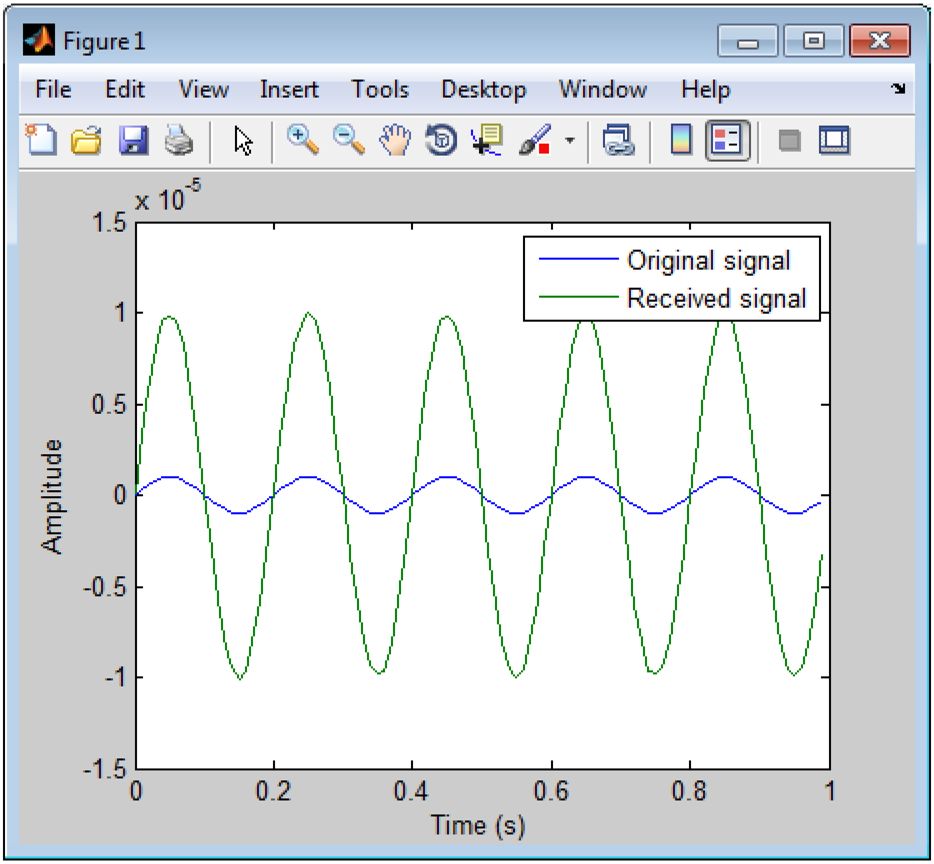
clone	Create receiver preamp object with same property values
getNumInputs	Number of expected inputs to step method
getNumOutputs	Number of outputs from step method
isLocked	Locked status for input attributes and nontunable properties
release	Allow property value and input characteristics changes
reset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

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`.

phased.ReceiverPreamp.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.ReceiverPreamp.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 `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.

---

phased.ReceiverPreamp.reset

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'`.

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 1-899.
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);
```

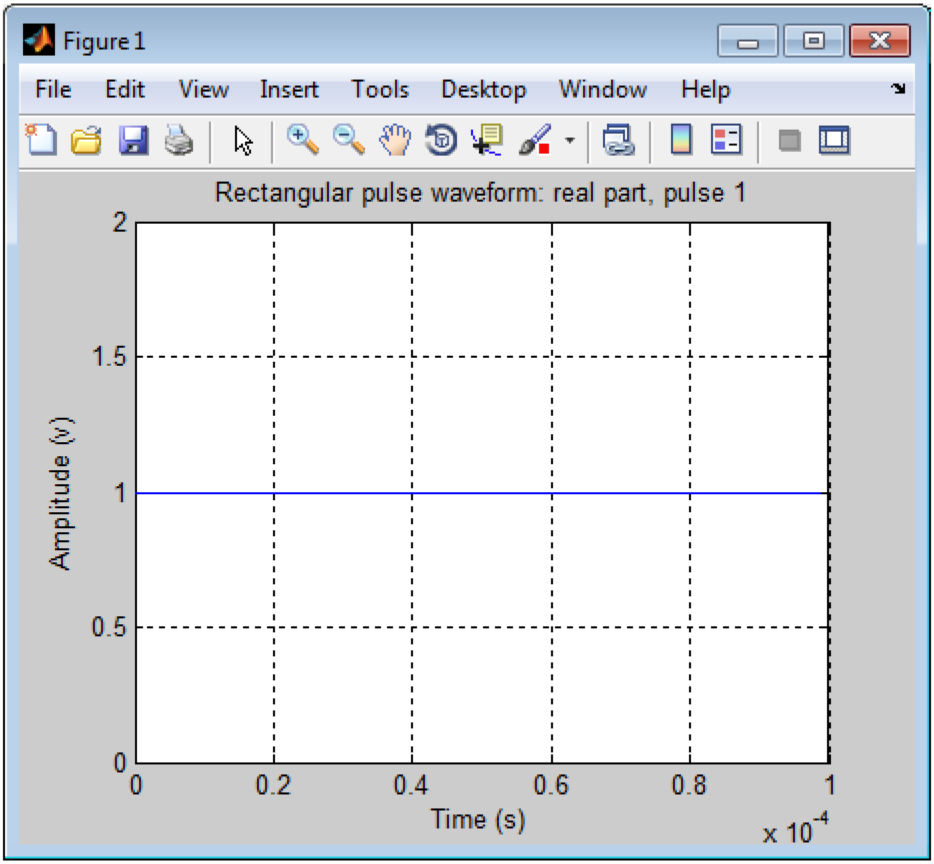


Figure 1
File Edit View Insert Tools Desktop Window Help
Rectangular pulse waveform: real part, pulse 1
Amplitude (v)
2
1.5
1
0.5
0
0
0.2
0.4
0.6
0.8
1
Time (s)
x 10^{-4}

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)
```

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`.

phased.RectangularWaveform.getMatchedFilter

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.

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.

phased.RectangularWaveform.plot

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 `PlotType` 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. `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);
```

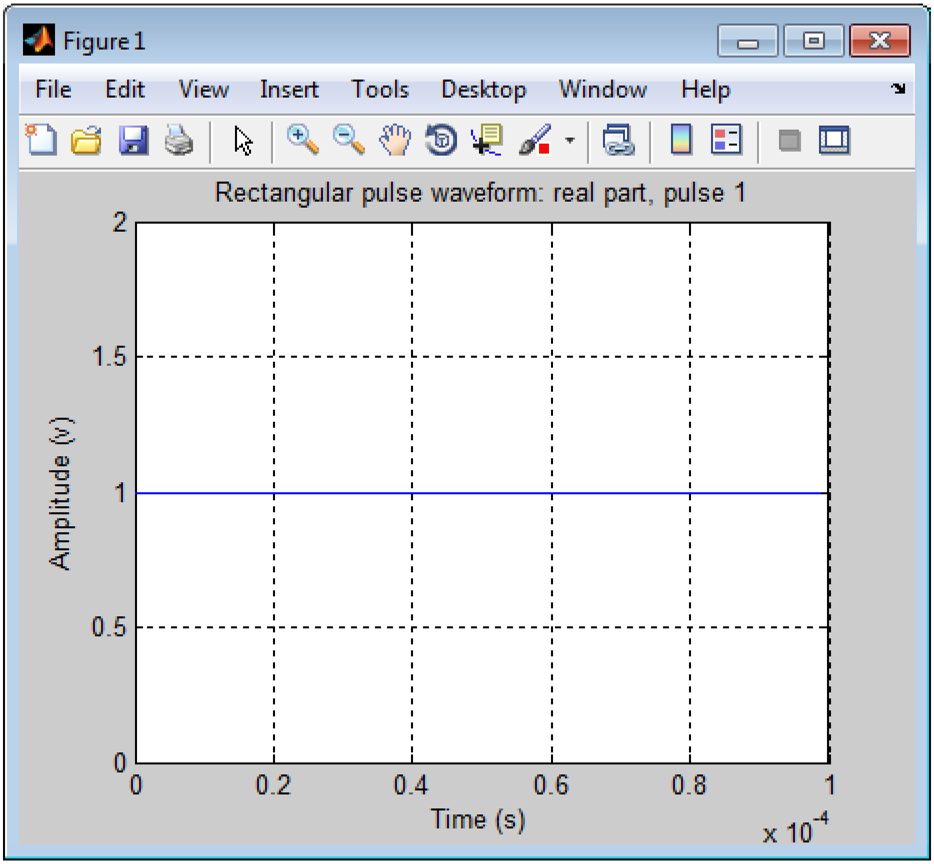


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Rectangular pulse waveform: real part, pulse 1
Amplitude (v)
Time (s)
x 10^-4
0
0.5
1
1.5
2
0
0.2
0.4
0.6
0.8
1

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.RectangularWaveform.reset

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 1-916.
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'`

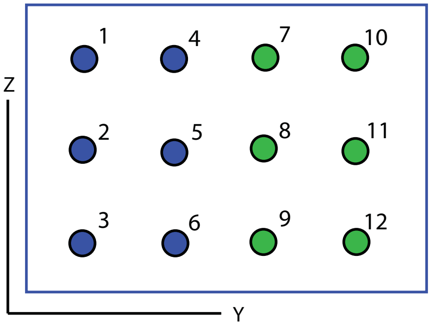
GridSize

Size of rectangular grid

Specify the size of the rectangular grid as a single positive integer or 1-by-2 positive integer row vector. This property applies only when you set the `Layout` property to `'Rectangular'`.

If `GridSize` is a scalar, the array has the same number of subarrays in each row and column.

If `GridSize` is a 1-by-2 vector, the vector has the form `[NumberOfRows, NumberOfColumns]`. The first entry is the number of subarrays along each column, while the second entry is the number of subarrays in each row. A row is along the local *y*-axis, and a column is along the local *z*-axis. This figure shows how a 3-by-2 URA subarray is replicated using a `GridSize` value of `[1,2]`.



3 x 2 Element URA
Replicated on a 1 x 2 Grid

Default: `[2 1]`

GridSpacing

Spacing of rectangular grid

Specify the rectangular grid spacing of subarrays as a real-valued positive scalar, a 1-by-2 row vector, or the string value `'Auto'`. This property applies only when you set the `Layout` property to `'Rectangular'`. Grid spacing units are expressed in meters.

If `GridSpacing` is a scalar, the spacing along the row and the spacing along the column is the same.

If `GridSpacing` is a length-2 row vector, it has the form `[SpacingBetweenRows, SpacingBetweenColumn]`. The first entry specifies the spacing between rows along a column. The second entry specifies the spacing between columns along a row.

If `GridSpacing` is `'Auto'`, the replication preserves the element spacing in both row and column. This option is available only if you use a `phased.ULA` or `phased.URA` object as the subarray.

Default: `'Auto'`

SubarrayPosition

Subarray positions in custom grid

Specify the positions of the subarrays in the custom grid. This property value is a 3-by-N matrix, where N indicates the number of subarrays in the array. Each column of the matrix represents the position of a single subarray in the array's local coordinate system, in meters, using the form [x; y; z].

This property applies when you set the `Layout` property to `'Custom'`.

Default: `[0 0; -0.5 0.5; 0 0]`

SubarrayNormal

Subarray normal directions in custom grid

Specify the normal directions of the subarrays in the array. This property value is a 2-by-N matrix, where N is the number of subarrays in the array. Each column of the matrix specifies the normal direction of the corresponding subarray, in the form [azimuth; elevation]. Each angle is in degrees and is defined in the local coordinate system.

You can use the `SubarrayPosition` and `SubarrayNormal` properties to represent any arrangement in which pairs of subarrays differ by certain transformations. The transformations can combine translation, azimuth rotation, and elevation rotation. However, you cannot use transformations that require rotation about the normal.

This property applies when you set the `Layout` property to `'Custom'`.

Default: `[0 0; 0 0]`

SubarraySteering

Subarray steering method

Specify the method of 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
isPolarizationCapable	Polarization capability
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',[1 2],...
   'GridSpacing','Auto');
```

Plot the azimuth response of the array. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

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

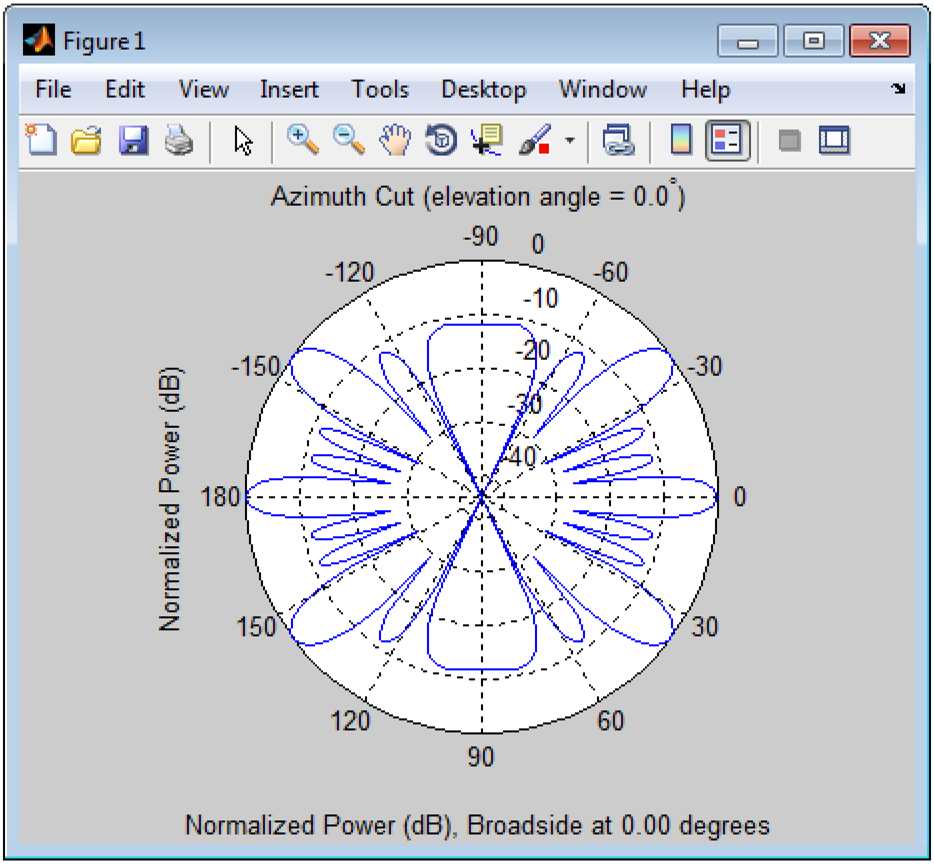


Figure 1
File Edit View Insert Tools Desktop Window Help
Azimuth Cut (elevation angle = 0.0°)
Normalized Power (dB)
-90
0
-120
-60
-10
-20
-30
-40
-150
-30
180
0
150
30
120
60
90
Normalized Power (dB), Broadside at 0.00 degrees

Response of Subarrays with Polarized Antenna Elements

Calculate the response at boresight for two 2-element ULAs that are subarrays of a 4-element ULA.

Create a two-element ULA of short-dipole antenna elements. Then, arrange two copies to form a 4-element ULA.

```
hsd = phased.ShortDipoleAntennaElement;
h = phased.ULA('Element',hsd,'NumElements',2,'ElementSpacing',0.5);
```

```
ha = phased.ReplicatedSubarray('Subarray',h,...
    'Layout','Rectangular','GridSize',[1 2],...
    'GridSpacing','Auto');
```

Find the response of each subarray at boresight. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

```
RESP = step(ha,1e9,[0;0],3e8)
```

```
RESP = 

    H: [2x1 double]
    V: [2x1 double]
```

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.

phased.ReplicatedSubarray.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.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`

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

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h - Replicated subarray

Replicated subarray specified as a `phased.ReplicatedSubarray` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability flag returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

Replicated Array of Short Dipoles Supports Polarization

Verify that a replicated subarray of `phased.ShortDipoleAntennaElement` short-dipole antenna elements supports polarization.

```
h = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[1e9 10e9]);
ha = phased.URA([3,2],'Element',h);
hr = phased.ReplicatedSubarray('Subarray',ha,...
    'Layout','Rectangular',...
  'GridSize',[1,2],'GridSpacing','Auto');
isPolarizationCapable(hr)
```

```
ans =

     1
```

The returned value `true (1)` shows that this array supports polarization.

phased.ReplicatedSubarray.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.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. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, then `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and `RespCut` is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the array response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where:

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'SteerAng'

Subarray steering angle. `SteerAng` can be either a 2-element column vector or a scalar.

If `SteerAng` is a 2-element column vector, it has the form [azimuth; elevation]. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

If `SteerAng` is a scalar, it specifies the azimuth angle. In this case, the elevation angle is assumed to be 0.

This option is applicable only if the `SubarraySteering` property of `H` is `'Phase'` or `'Time'`.

Default: `[0;0]`

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'Weights'

Weight values applied to the array, specified as a length-N column vector or N-by-M matrix. The dimension N is the number of subarrays in the array. The interpretation of M depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimension	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-M row vector	Apply one set of weights for the same single frequency or all M frequencies.
N-by-M matrix	Scalar	Apply all of the M different columns in `Weights` for the same single frequency.
	1-by-M row vector	Apply each of the M different columns in `Weights` for the corresponding frequency in `FREQ`.

'AzimuthAngles'

Azimuth angles for plotting subarray response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting subarray response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting subarray response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting subarray response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Azimuth Response and Directivity of ULA with Subarrays

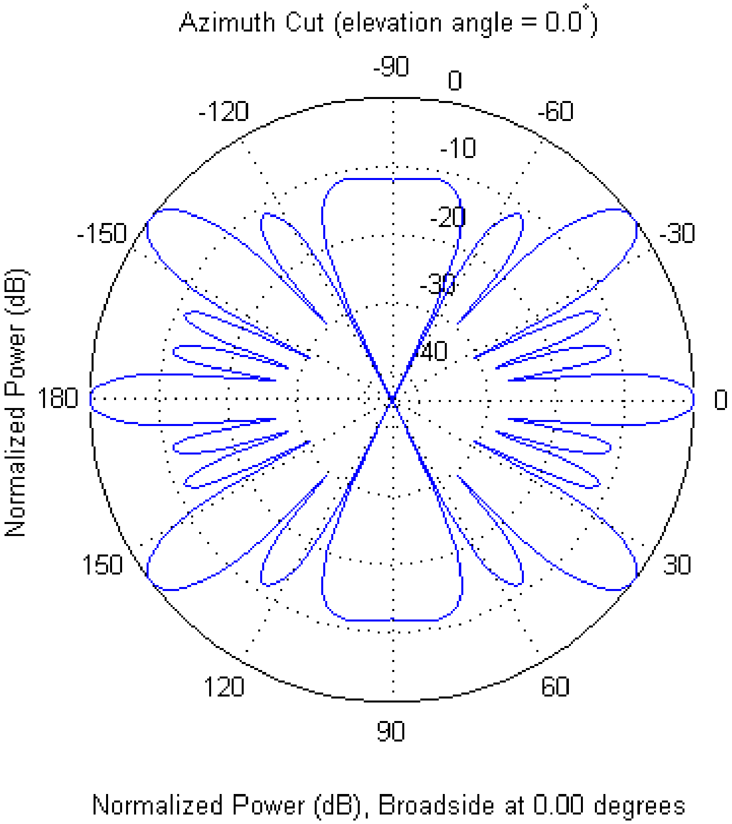
Plot the azimuth response of a 4-element ULA composed of two 2-element ULAs.

Create a 2-element ULA, and arrange two copies to form a 4-element ULA.

```
h = phased.ULA('NumElements',2,'ElementSpacing',0.5);
ha = phased.ReplicatedSubarray('Subarray',h,...
    'Layout','Rectangular','GridSize',[1 2],...
    'GridSpacing','Auto');
```

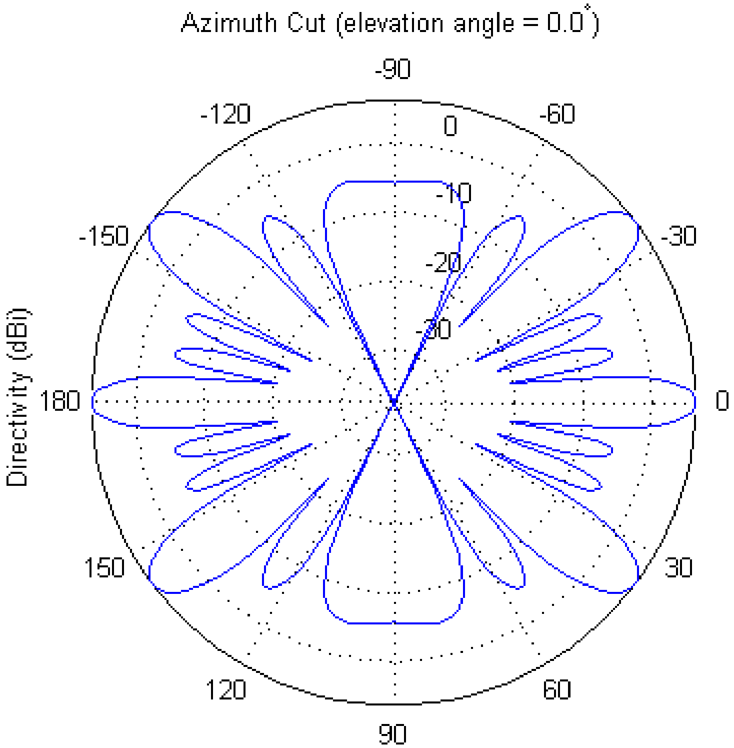
Plot the azimuth response of the array. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

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



Plot the azimuth directivity of the array.

```
plotResponse(ha,1e9,3e8,'RespCut','Az','Format','Polar','Unit','dbi')
```

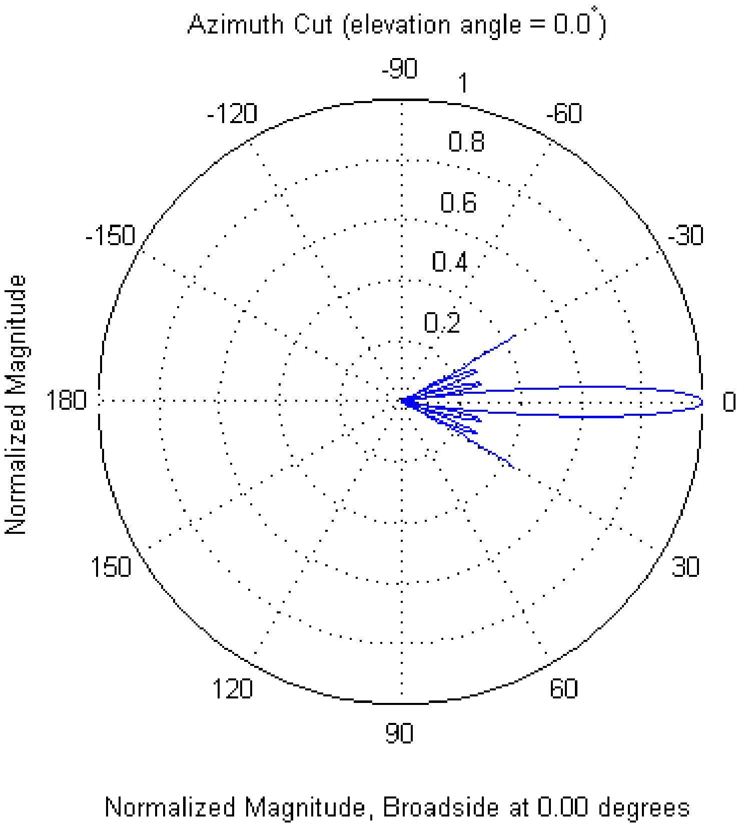


Directivity (dBi), Broadside at 0.00 degrees

Display Azimuth Response of Array with Subarrays Between -30 and 30 Degrees

Create a 2-element ULA, and arrange two copies to form a 4-element ULA. Using the `AzimuthAngles` parameter, plot the response within a restricted range of azimuth angles from -30 to 30 degrees in 0.1 degree increments.

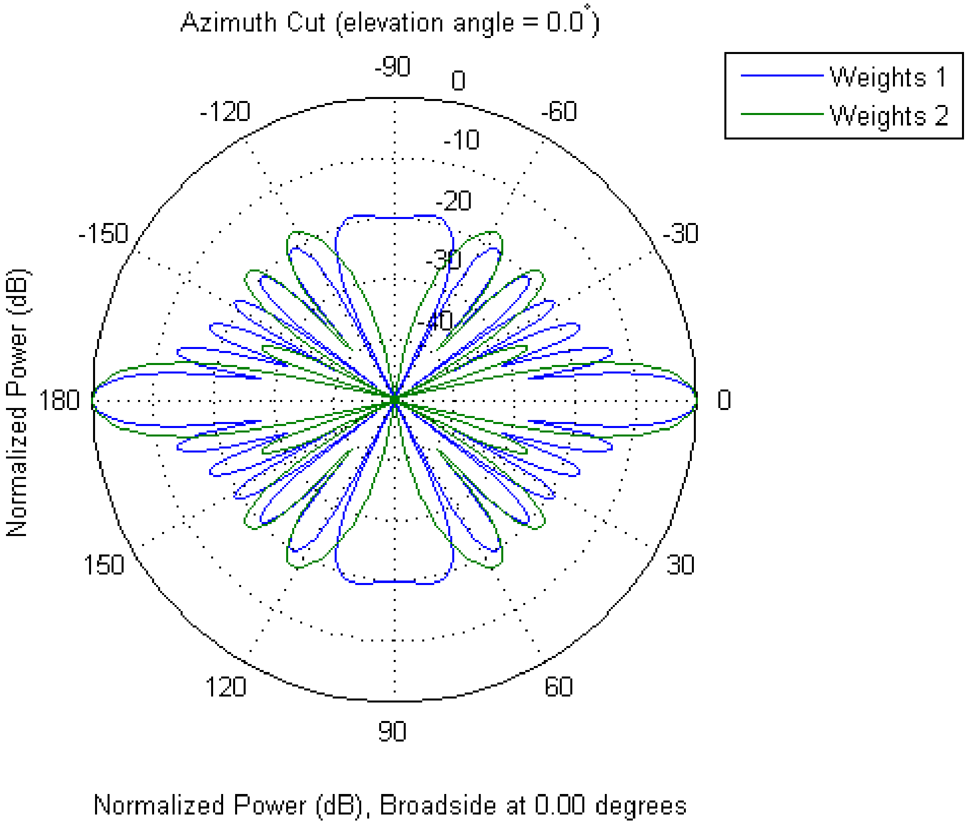
```
h = phased.ULA('NumElements',2,'ElementSpacing',0.5);
ha = phased.ReplicatedSubarray('Subarray',h,...
    'Layout','Rectangular','GridSize',[1 2],...
    'GridSpacing','Auto');
plotResponse(ha,1e9,3e8,'RespCut','Az','Format','Polar',...
    'AzimuthAngles',[-30:0.1:30],'Unit','mag');
```



Apply Two Sets of Weights at a Single Frequency

Construct an array of replicated subarrays. Start with a 2-element uniform line array (ULA), and duplicate it 5 times to create a 10-element ULA. Apply both uniform weights and tapered weights. Then, use `plotResponse` to show that the tapered set of weights reduces the adjacent sidelobes while broadening the main lobe.

```
h = phased.ULA('NumElements',2,'ElementSpacing',0.2);
ha = phased.ReplicatedSubarray('Subarray',h,...
    'Layout','Rectangular','GridSize',[1 5],...
    'GridSpacing',0.4);
c = physconst('LightSpeed');
fc = 1e9;
wts1 = [0.2,0.2,0.2,0.2,0.2]';
wts2 = [0.1,0.23333,.33333,0.23333,0.1]';
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar',...
    'Weights',[wts1,wts2]);
```



See Also `uv2azel` | `azel2uv`

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.

---

phased.ReplicatedSubarray.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`. `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

Voltage responses of the subarrays of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions N-by-M-by-L. The first dimension, N , represents the number of subarrays in the phased array, the second dimension, M, represents the number of angles specified in `ANG`, while L represents the number of frequencies specified in `FREQ`. Each column of `RESP` contains the responses of the subarrays for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP` contains the responses of the subarrays for the corresponding frequency specified in `FREQ`.

- If the array is capable of supporting polarization, the voltage response, `RESP`, is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`, each having dimensions *N*-by-*M*-by-*L*. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. The first dimension, *N* , represents the number of subarrays in the phased array, the second dimension, *M*, represents the number of angles specified in `ANG`, while *L* represents the number of frequencies specified in `FREQ`. Each of the *M* columns contains the responses of the subarrays for the corresponding direction specified in `ANG`. Each of the *L* pages contains the responses of the subarrays for the corresponding frequency specified in `FREQ`.

Examples

Response of Subarrays

Calculate the response at boresight for two 2-element ULA's that are subarrays of a 4-element ULA of short-dipole antenna elements.

Create a two-element ULA of short-dipole antenna elements. Then, arrange two copies to form a 4-element ULA.

```
hsd = phased.ShortDipoleAntennaElement;
h = phased.ULA('Element',hsd,'NumElements',2,'ElementSpacing',0.5);
ha = phased.ReplicatedSubarray('Subarray',h,...
    'Layout','Rectangular','GridSize',[1 2],...
    'GridSpacing','Auto');
```

Find the response of each subarray at boresight. Assume the operating frequency is 1 GHz and the wave propagation speed is 3e8 m/s.

```
RESP = step(ha,1e9,[0;0],3e8)
```

```
RESP =
    H: [2x1 double]
    V: [2x1 double]
```

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. `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`

'ShowTaper'

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

'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)
```

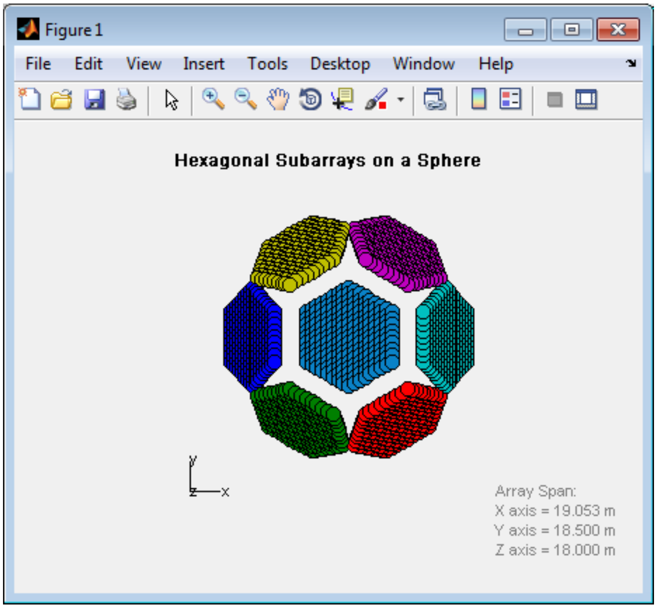


Figure 1
File Edit View Insert Tools Desktop Window Help
Hexagonal Subarrays on a Sphere
y
z
x
Array Span:
X axis = 19.053 m
Y axis = 18.500 m
Z axis = 18.000 m

See Also `phased.ArrayResponse`

Related Examples

- Phased Array Gallery

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 1-957.
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

`broadside2az` | `phased.RootWSFEstimator`

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.

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.RootMUSICEstimator.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.RootMUSICEstimator('SensorArray',ha,...
    'OperatingFrequency',fc,...
    'NumSignalsSource','Property','NumSignals',2);
doas = step(hdoa,x+noise);
az = broadside2az(sort(doas),[20 60])
```

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 1-967.

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

`broadside2az` | `phased.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`.

phased.RootWSFEstimator.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.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.

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])
```

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 1-977.
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

Sensor array

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

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

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);
```

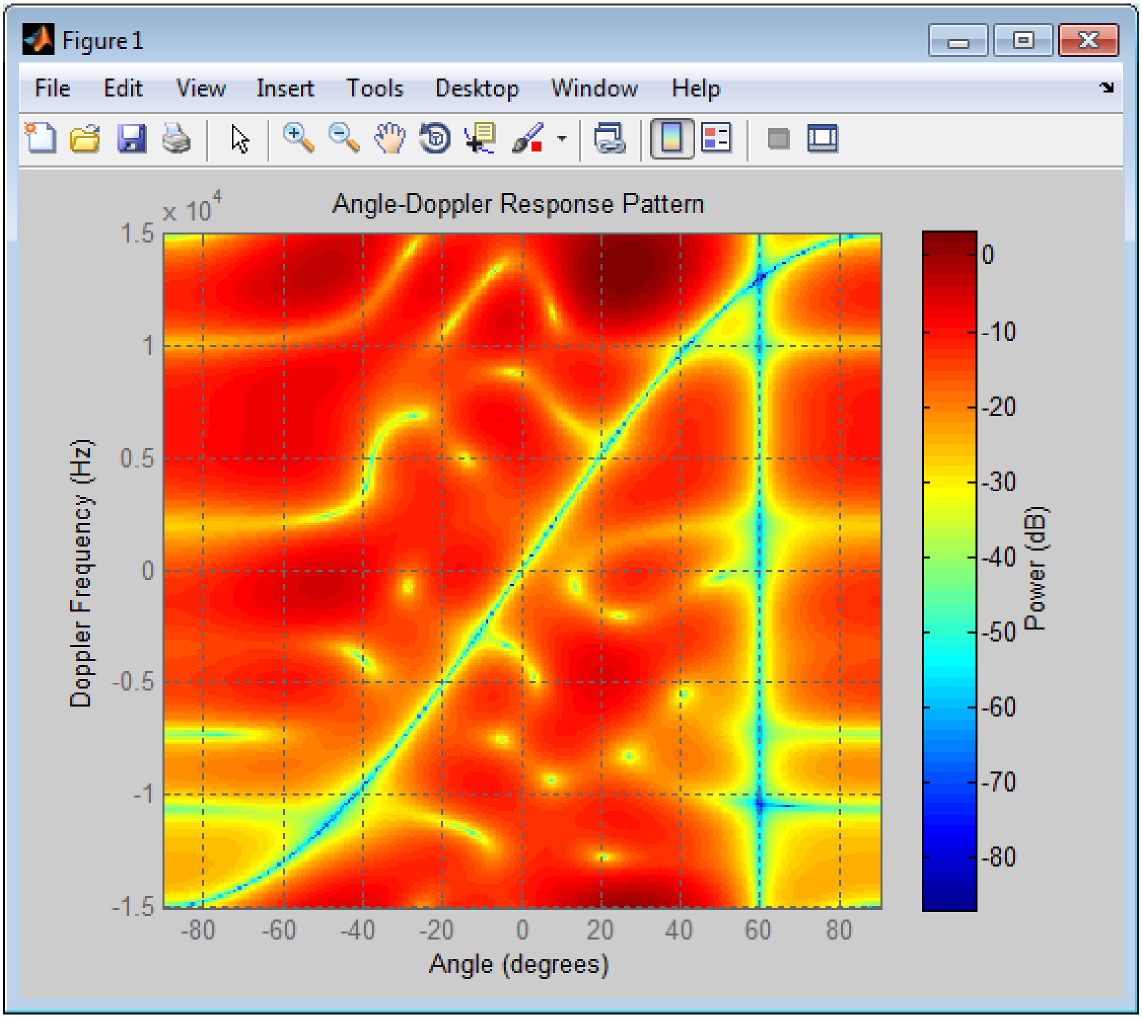


Figure 1
File Edit View Insert Tools Desktop Window Help
Angle-Doppler Response Pattern
x 10^4
1.5
1
0.5
0
-0.5
-1
-1.5
Doppler Frequency (Hz)
-80 -60 -40 -20 0 20 40 60 80
Angle (degrees)
0
-10
-20
-30
-40
-50
-60
-70
-80
Power (dB)

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`

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.

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.STAPSMIBeamformer.step

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

Short-dipole antenna element

Description

The `phased.ShortDipoleAntennaElement` object models a short-dipole antenna element. A short-dipole antenna is a center-fed length wire whose length is much shorter than a wavelength. This antenna object only supports polarized fields.

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

1. Define and set up your short-dipole antenna element. See "Construction" on page 1-992 .
2. Call `step` to compute the antenna response according to the properties of `phased.ShortDipoleAntennaElement`. The behavior of `step` is specific to each object in the toolbox.

Construction

`h = phased.ShortDipoleAntennaElement` creates the system object, `h`, to model a short-dipole antenna element.

`h = phased.ShortDipoleAntennaElement(Name,Value)` creates the system 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

Antenna operating frequency range

Antenna operating frequency range specified as a 1-by-2 row vector in the form of `[LowerBound HigherBound]`. This vector defines the frequency range over which the antenna has a response. The antenna element has no response outside the specified frequency range.

Default: `[0 1e20]`

AxisDirection

Dipole axis direction

Dipole axis direction specified as one of `'Y'` or `'Z'`. This axis defines the direction of the dipole current with respect to the local coordinate system. In this coordinate system, the *x*-axis corresponds to the boresight direction. Two dipole axis directions are allowed: `'Y'` specifies a horizontal dipole and `'Z'` specifies a vertical dipole in the local coordinate system.

Default: `'Z'`

Methods

clone	Create short-dipole 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
isPolarizationCapable	Polarization capability
plotResponse	Plot response pattern of antenna
release	Allow property value and input characteristics changes
step	Output response of antenna element

Examples

Short-dipole Antenna Aligned Along the Y-Axis

Specify a short-dipole antenna with its dipole oriented along the *y*-axis. Then, plot the 3-D responses for both the horizontal and vertical polarizations.

```
h1 = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6,600e6],'AxisDirection','Y');
fc = 250e6;
```

```
figure;
plotResponse(h1,fc,'Format','Polar',...
    'RespCut','3D','Polarization','H');
figure;
plotResponse(h1,fc,'Format','Polar',...
    'RespCut','3D','Polarization','V');
figure;
plotResponse(h1,fc,'Format','Polar',...
    'RespCut','3D','Polarization','C');
```

This figure shows the horizontal polarization response.

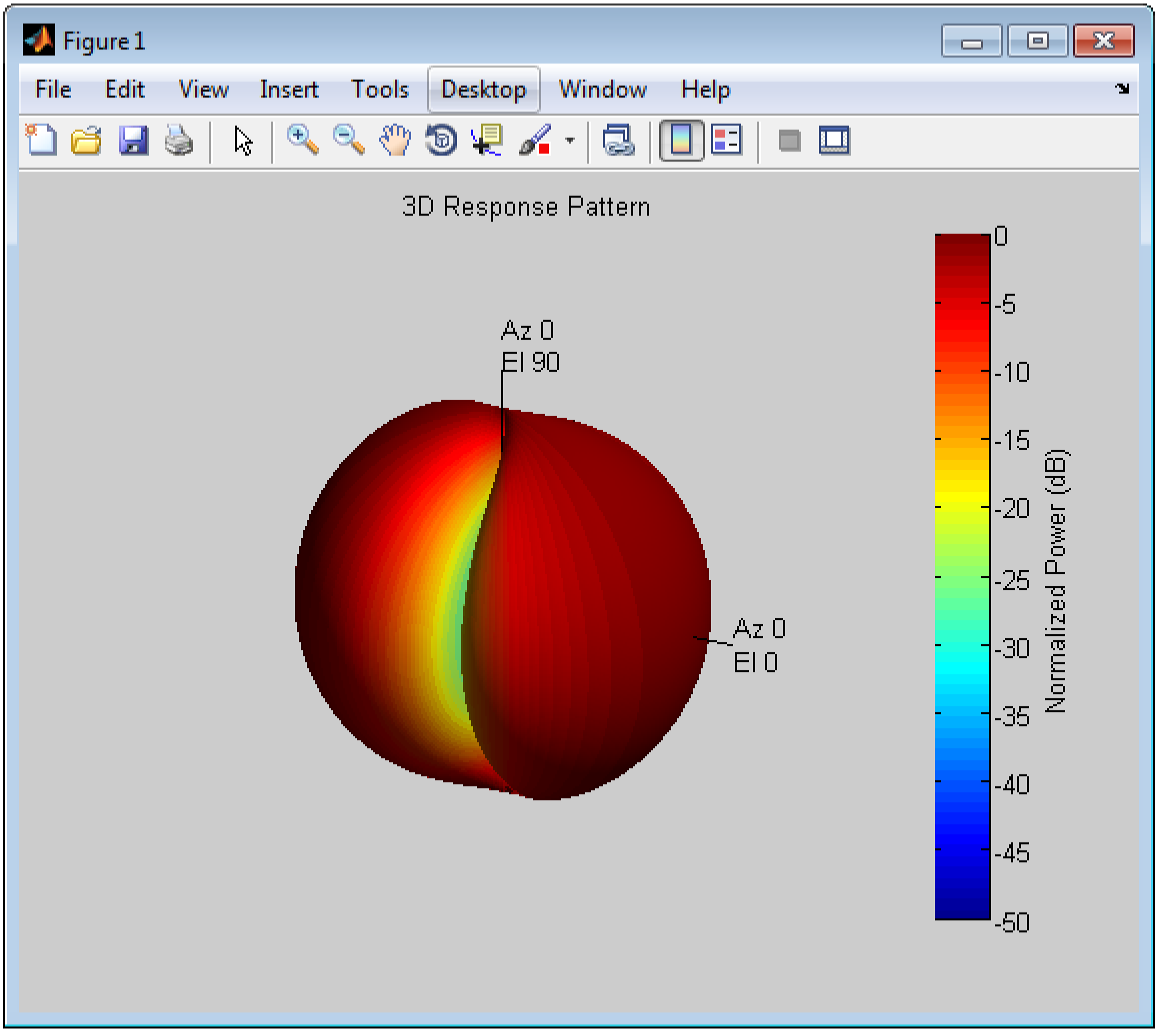


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
3D Response Pattern
Az 0
El 90
Az 0
El 0
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
-50
Normalized Power (dB)

This figure shows the vertical polarization response.

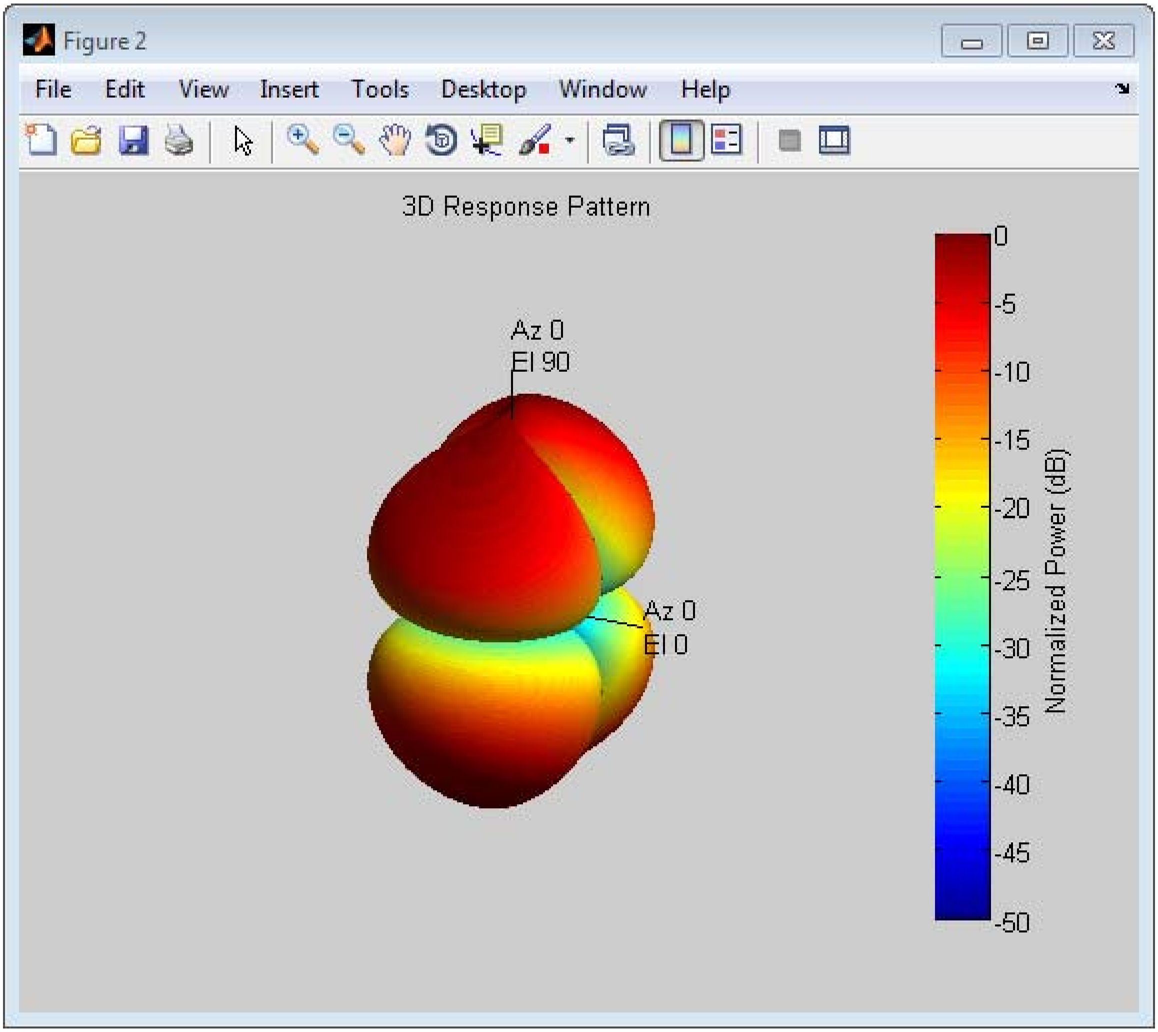


Figure 2
File Edit View Insert Tools Desktop Window Help
3D Response Pattern
Az 0
El 90
Az 0
El 0
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
-50
Normalized Power (dB)

This combined response best illustrates the polarity of the short-dipole.

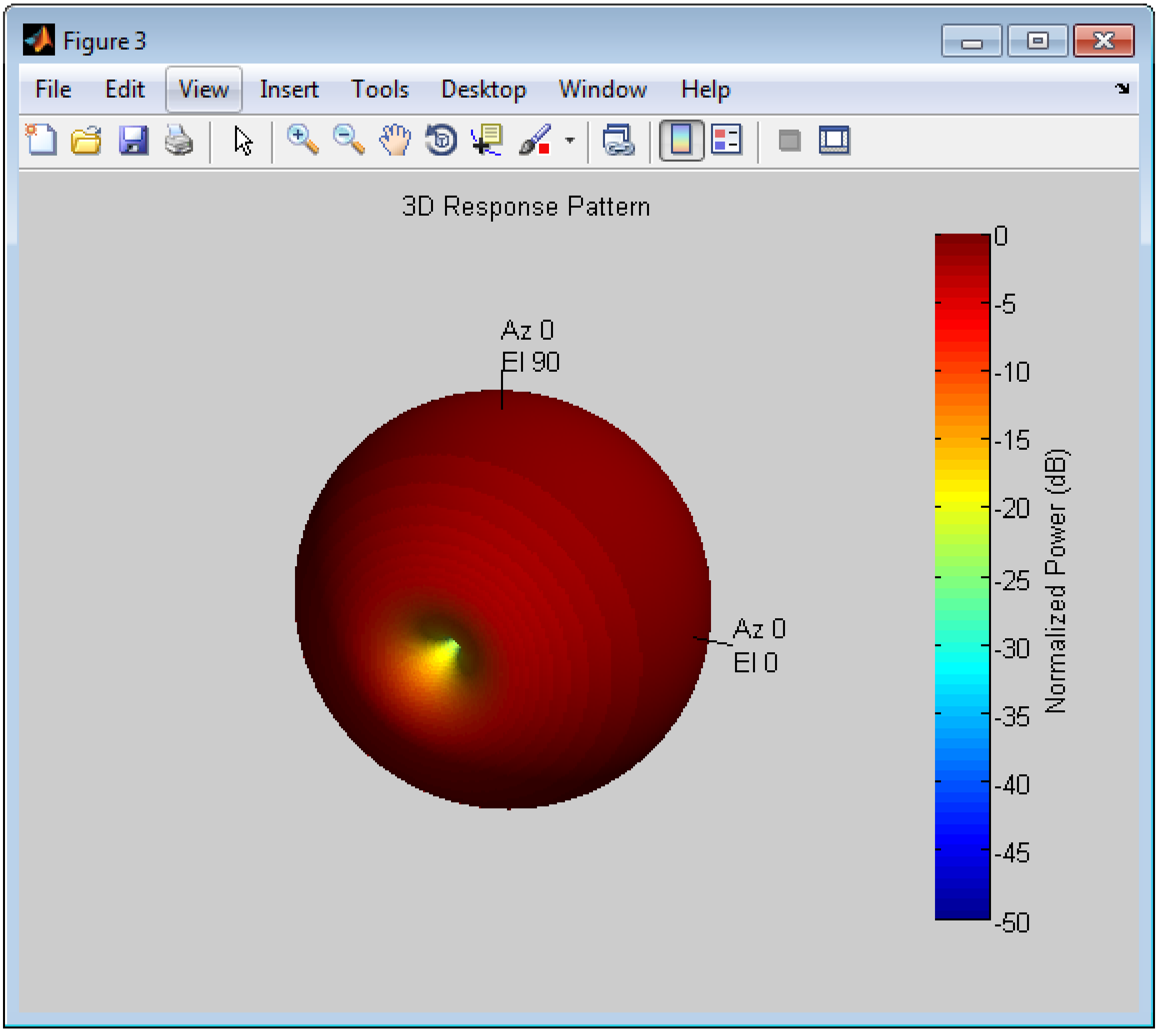


Figure 3
File
Edit
View
Insert
Tools
Desktop
Window
Help
3D Response Pattern
Az 0
El 90
Az 0
El 0
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
-50
Normalized Power (dB)

Algorithms

The total response of a short-dipole antenna element is a combination of its frequency response and spatial response. This System object calculates both responses using nearest neighbor interpolation and then multiplies the responses to form the total response.

References

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

See Also

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

phased.ShortDipoleAntennaElement.clone

Purpose

Create short-dipole 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.ShortDipoleAntennaElement.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.ShortDipoleAntennaElement.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.ShortDipoleAntennaElement.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 `phased.ShortDipoleAntennaElement` 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

Polarization capability

Syntax

```
flag = isPolarizationCapable(h)
```

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the `phased.ShortDipoleAntennaElement` antenna element supports polarization or not. An antenna element supports polarization if it can create or respond to polarized fields. The `phased.ShortDipoleAntennaElement` object always supports polarization.

Input Arguments

h - Short-dipole antenna element

Short-dipole antenna element specified as a `phased.ShortDipoleAntennaElement` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability returned as a Boolean value `true` if the antenna element supports polarization or `false` if it does not. Because the short-dipole antenna element supports polarization, the returned value is always `true`.

Examples

Short-Dipole Antenna Supports Polarization

Determine whether a `phased.ShortDipoleAntennaElement` antenna element supports polarization.

```
h = phased.ShortDipoleAntennaElement;
isPolarizationCapable(h)
```

```
ans =

     1
```

The returned value `true (1)` shows that this antenna element supports polarization.

phased.ShortDipoleAntennaElement.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 System object

FREQ

Operating frequency in Hertz specified as a scalar or 1–by-*K* row vector. `FREQ` must lie within the range specified by the `FrequencyVector` property of `H`. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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 specified as a scalar. This argument is applicable only when `RespCut` is `'Az'` or `'El'`. If `RespCut` is `'Az'`, `CutAngle` must be between –90 and 90. If `RespCut` is `'El'`, `CutAngle` must be between –180 and 180.

Default: `0`

'Format'

Format of the plot, using one of `'Line'`, `'Polar'`, or `'UV'`. If you set `Format` to `'UV'`, `FREQ` must be a scalar.

Default: `'Line'`

'NormalizeResponse'

Set this value to `true` to normalize the response pattern. Set this value to `false` to plot the response pattern without normalizing it. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the antenna response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For antennas that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'AzimuthAngles'

Azimuth angles for plotting element response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting element response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting element response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting element response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

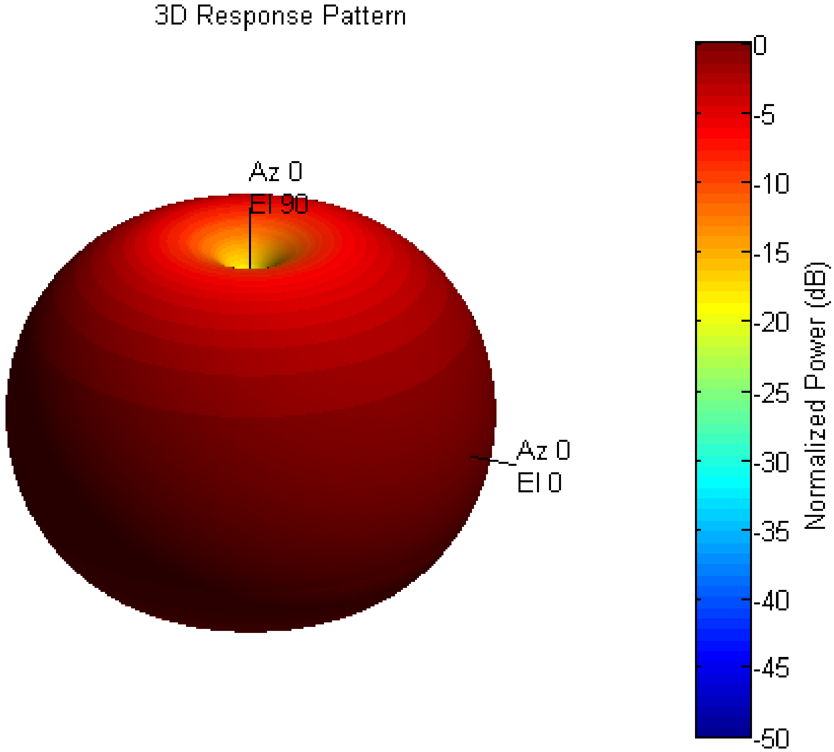
Response of Short-Dipole Antenna Oriented Along the Z-Axis

Specify a short-dipole antenna element with its dipole axis pointing along the z-axis. To do so, set the `'AxisDirection'` value to `'Z'`.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Z');
```

Plot the antenna's vertical polarization response at 200 MHz as a 3-D polar plot.

```
fc = 200e6;
plotResponse(sSD,fc,'Format','Polar',...
    'RespCut','3D','Polarization','V');
```



As the above figure shows, the antenna pattern is that of a vertically-oriented dipole and has its maximum at the equator and nulls at the poles.

Plot Short-Dipole Antenna Element Response Over Selected Range

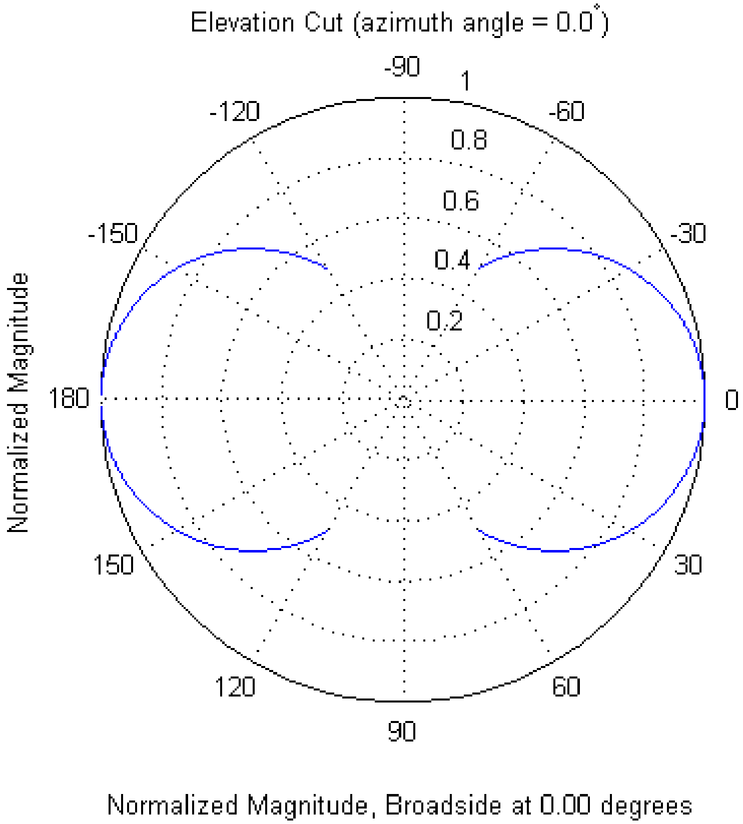
This example shows how to construct a short-dipole antenna element with its dipole axis pointing along the z-axis and how to plot the response over a selected range of angles. The antenna operating frequency spans the range 100 to 900 MHz.

To construct a z-directed short-dipole antenna, set the `'AxisDirection'` value to `'Z'`.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Z');
```

Plot the antenna's vertical polarization response at 200 MHz as an elevation cut at a fixed azimuth angle. Use the 'ElevationAngles' property to restrict the plot from -60 to 60 degrees elevation in 0.1 degree increments.

```
plotResponse(sSD,200e6,'Format','Polar',...
    'RespCut','El','Polarization','V',...
    'ElevationAngles',[-60:0.1:60],'Unit','mag');
```



Plot Short-Dipole Antenna Element Directivity

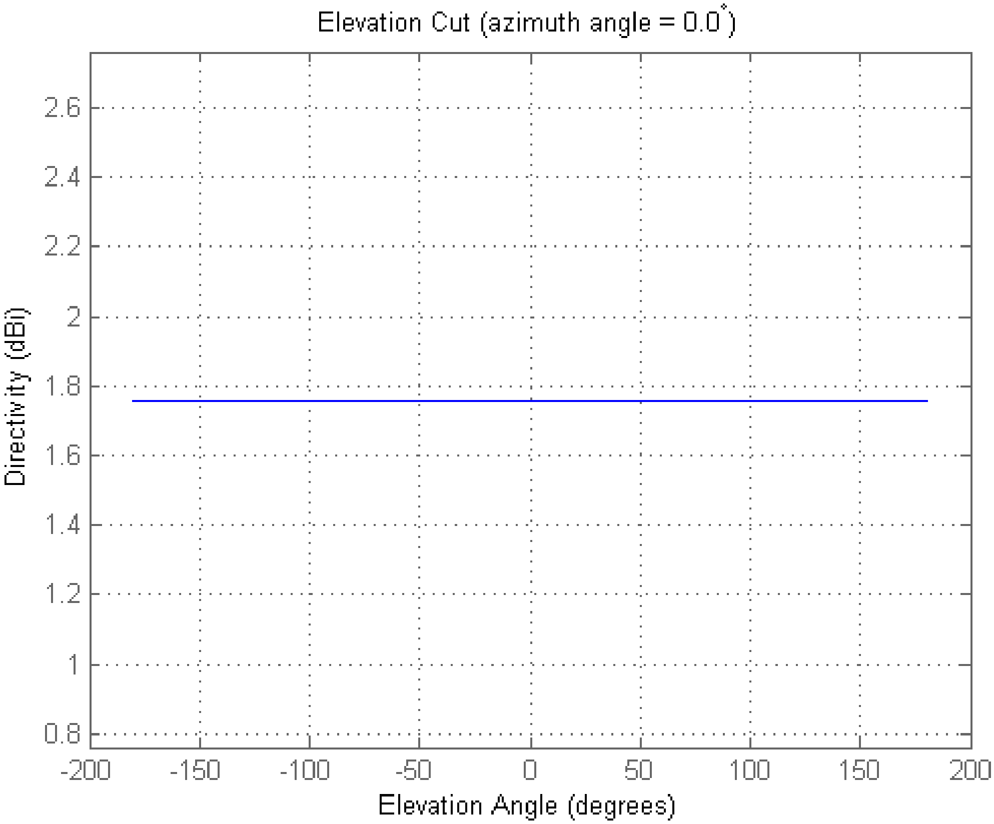
This example shows how to construct a short-dipole antenna element with its dipole axis pointing along the y-axis and how to plot the directivity. The antenna operating frequency spans the range 100 to 900 MHz.

To construct a y-directed short-dipole antenna, set the `'AxisDirection'` value to `'Y'`.

```
sSD = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Y');
```

Plot the antenna's directivity at 500 MHz as an elevation cut at a fixed azimuth angle.

```
plotResponse(sSD,500e6,'Format','Line',...
    'RespCut','El','Unit','dbi');
```



See Also

`uv2azel` | `azel2uv`

phased.ShortDipoleAntennaElement.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 the operating frequencies specified in `FREQ` and in the directions specified in `ANG`. For the short-dipole antenna element object, `RESP` is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`, representing the horizontal and vertical polarization components of the antenna's response. Each field is an M-by-L matrix containing the antenna response at the M angles specified in `ANG` and at the L frequencies specified in `FREQ`.

Note The object performs an initialization the first time the `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 returned as a MATLAB `struct` with fields `RESP.H` and `RESP.V`. Both `RESP.H` and `RESP.V` contain responses for the horizontal and vertical polarization components of the antenna radiation pattern. Both `RESP.H` and `RESP.V` are M-by-L matrices. In these matrices, M represents the number of angles specified in `ANG`, and L represents the number of frequencies specified in `FREQ`.

Examples

Find the response of a short-dipole antenna element at the boresight angle, `[0;0]`, and at off-boresight, `[30;0]`. The antenna operates between 100 and 900 MHz. Compute the response of the antenna at these angles.

```
hsd = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100 900]*1e6,'AxisDirection','Y');
ang = [0 30;0 0];
fc = 250e6;
resp = step(hsd,fc,ang);
```

```
resp =

    H: [2x1 double]
    V: [2x1 double]
```

Algorithms

The total response of a short-dipole antenna element is a combination of its frequency response and spatial response. This System object calculates both responses using nearest neighbor interpolation and then multiplies the responses to form the total response.

See Also `uv2azel` | `phitheta2azel`

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 1-1018.
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`

EnablePolarization

Enable polarization simulation

Set to this property to `true`, to enable the steering vector to simulate polarization. Set this property to `false` to ignore polarization. This property applies only when the array specified in the `SensorArray` property is capable of simulating polarization and you have set the `IncludeElementResponse` property to `true`.

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 and display the steering vector for a 4-element uniform linear array in the direction of 30 degrees azimuth and 20 degrees elevation. Assume the array's operating frequency is 300 MHz.

```
hULA = phased.ULA('NumElements',4);
hsv = phased.SteeringVector('SensorArray',hULA);
Fc = 3e8;
ANG = [30; 20];
sv = step(hsv,Fc,ANG)
```

```
sv =

  -0.6011 - 0.7992i
   0.7394 - 0.6732i
   0.7394 + 0.6732i
  -0.6011 + 0.7992i
```

Beam Pattern With and Without Steering

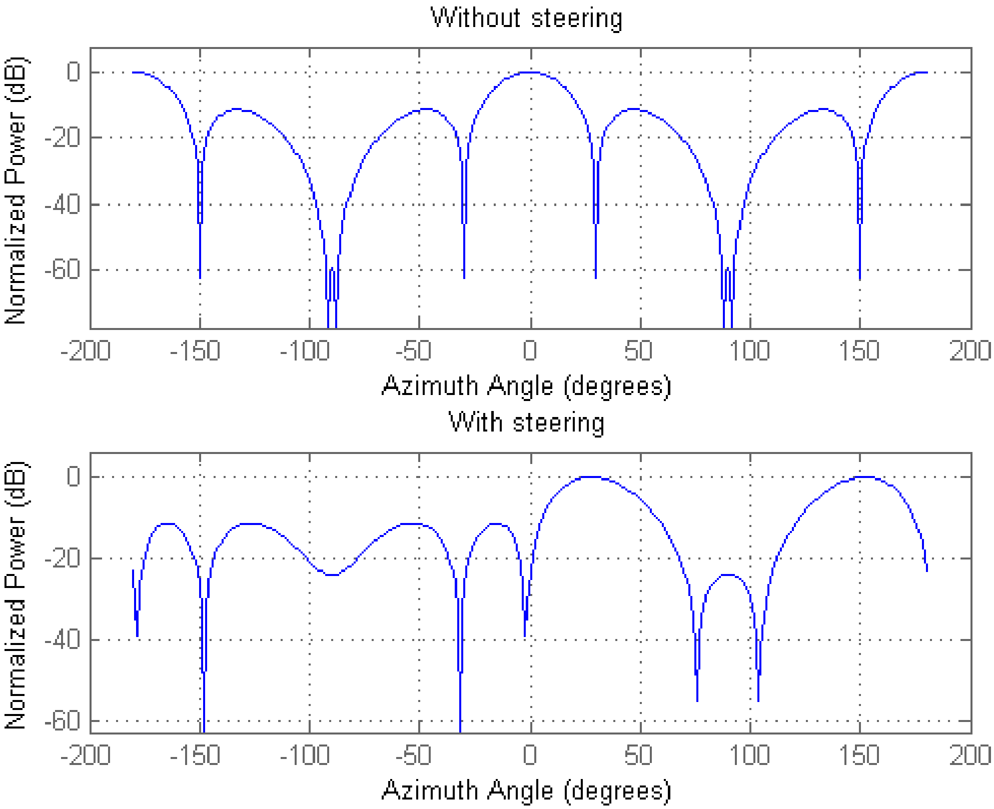
Calculate the steering vector for a 4-element uniform linear array in the direction of 30 degrees azimuth and 20 degrees elevation. Assume the array's operating frequency is 300 MHz.

```
fc = 3e8;
ha = phased.ULA('NumElements',4);
hsv = phased.SteeringVector('SensorArray',ha);
```

```
sv = step(hsv,fc,[30; 20]);
```

Plot the beam patterns for the uniform linear array when no steering vector is applied (steered broadside) and when a steering vector is applied.

```
c = hsv.PropagationSpeed;
subplot(211)
plotResponse(ha,fc,c,'RespCut','Az');
title('Without steering');
subplot(212)
plotResponse(ha,fc,c,'RespCut','Az','Weights',sv);
title('With 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.

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.

phased.SteeringVector.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.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. The form of the steering vector depends upon whether the `EnablePolarization` property is set to `true` or `false`.

- If `EnablePolarization` is set to `false`, the steering vector, `SV`, has the dimensions N-by-M-by-L. The first dimension, *N*, is the number of elements of the phased array or, if `H.SensorArray` contains subarrays, the number of subarrays. Each column of `SV` contains the steering vector of the array for the corresponding direction specified in `ANG`. Each of the *L* pages of `SV` contains the steering vectors of the array for the corresponding frequency specified in `FREQ`.

If you set the `H.IncludeElementResponse` property to `true`, the steering vector includes the individual element responses. If you set the `H.IncludeElementResponse` property to `false`, the elements are assumed to be isotropic. Then, the steering vector does not include individual element responses.

- If `EnablePolarization` is set to `true`, `SV` is a MATLAB `struct` containing two fields, `SV.H` and `SV.V`. These fields represent the steering vector's horizontal and vertical polarization components. Each field has the dimensions *N*-by-*M*-by-*L*. The first dimension, *N*, is the number of elements of the phased array or, if `H.SensorArray` contains subarrays, the number of subarrays. Each column of `SV` contains the steering vector of the array for the corresponding direction specified in `ANG`. Each of the *L* pages of `SV` contains the steering vectors of the array for the corresponding frequency specified in `FREQ`.

  If you set the `EnablePolarization` to `false` for an array that supports polarization, then all polarization information is discarded. The combined pattern from both `H` and `V` polarizations is used at each element to compute the steering vector.

  Simulating polarization also requires that the sensor array specified in the `SensorArray` property is capable of simulating polarization, and the `IncludeElementResponse` property is set to `true`.

Examples

Steering Vector for Uniform Linear Array

Calculate the steering vector for a uniform linear array 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`

phased.SteppedFMWaveform

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 1-1032.
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);
```

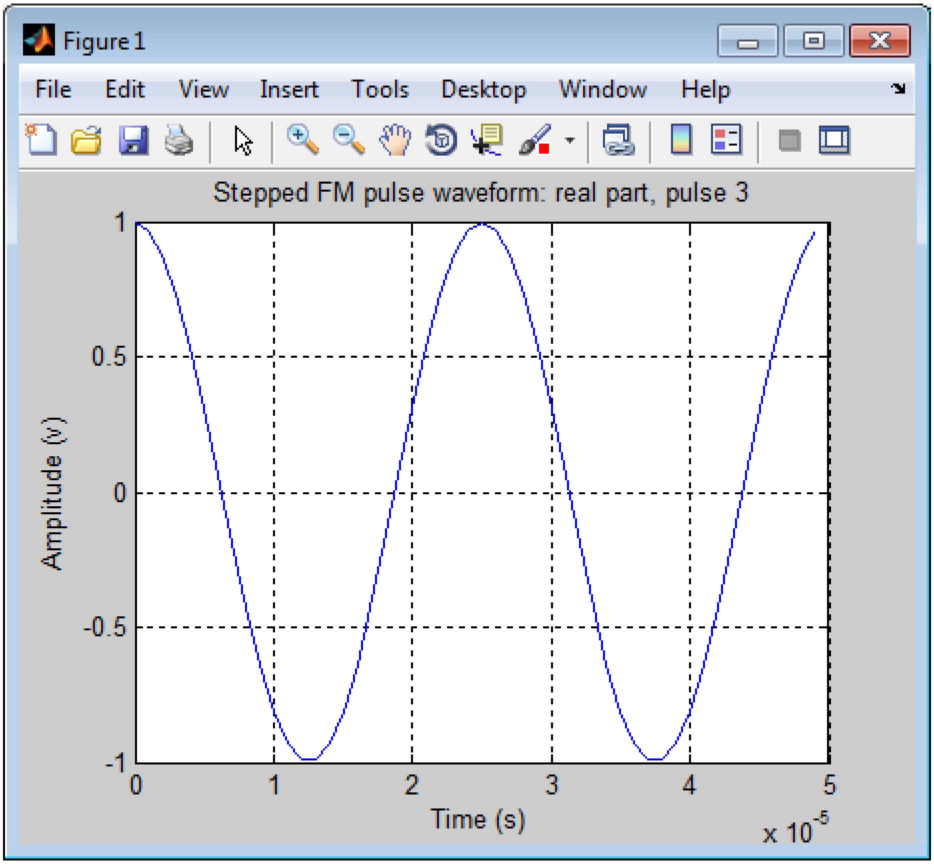


Figure 1
File Edit View Insert Tools Desktop Window Help
Stepped FM pulse waveform: real part, pulse 3
Amplitude (v)
1
0.5
0
-0.5
-1
0
1
2
3
4
5
Time (s)
x 10^{-5}

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)
```

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`.

phased.SteppedFMWaveform.getMatchedFilter

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.

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.

phased.SteppedFMWaveform.plot

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 `PlotType` 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. `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);
```

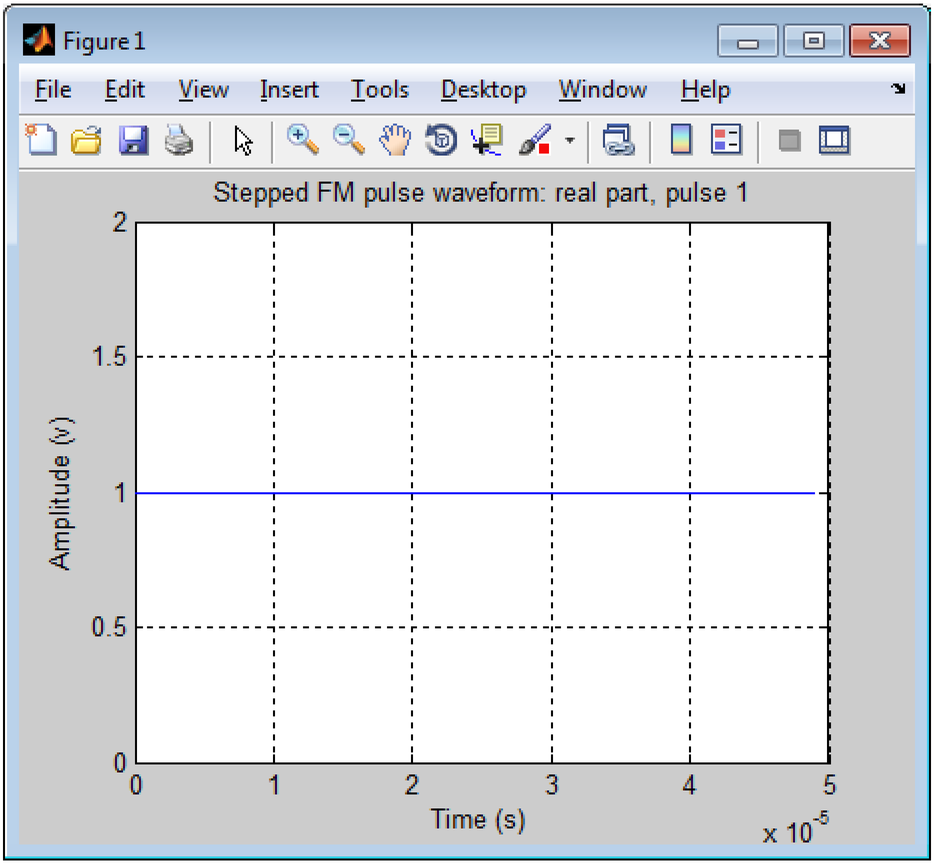


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Stepped FM pulse waveform: real part, pulse 1
Amplitude (v)
2
1.5
1
0.5
0
0
1
2
3
4
5
Time (s)
x 10-5

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.

---

phased.SteppedFMWaveform.reset

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.

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);
```

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 1-1051.
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.

```
[Pxx,F] = periodogram(y,[],2048,hs.SampleRate,'centered');
plot(F/1000,10*log10(Pxx)); grid;
xlabel('Frequency (kHz)');
```

```
ylabel('Power/Frequency (dB/Hz)');
title('Periodogram Power Spectrum Density Estimate');
```

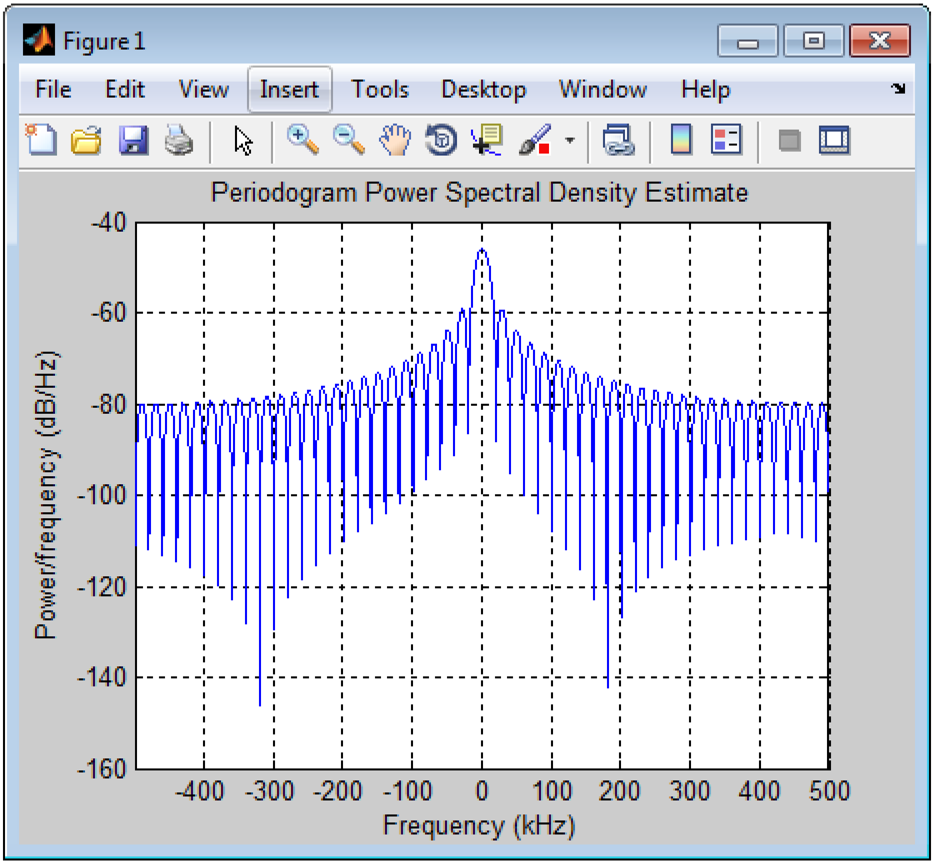


Figure 1
File Edit View Insert Tools Desktop Window Help
Periodogram Power Spectral Density Estimate
-40
-60
-80
-100
-120
-140
-160
Power/frequency (dB/Hz)
-400 -300 -200 -100 0 100 200 300 400 500
Frequency (kHz)

Detect the range.

```
[~,rngidx] = findpeaks(pow2db(Pxx/max(Pxx)),...
    'MinPeakHeight',-5);
rngfreq = F(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.

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.

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.StretchProcessor.step

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.

```
[Pxx,F] = periodogram(y,[],2048,hs.SampleRate,'centered');
```

```
plot(F/1000,10*log10(Pxx)); grid;
xlabel('Frequency (kHz)');
ylabel('Power/Frequency (dB/Hz)');
title('Periodogram Power Spectrum Density Estimate');
```

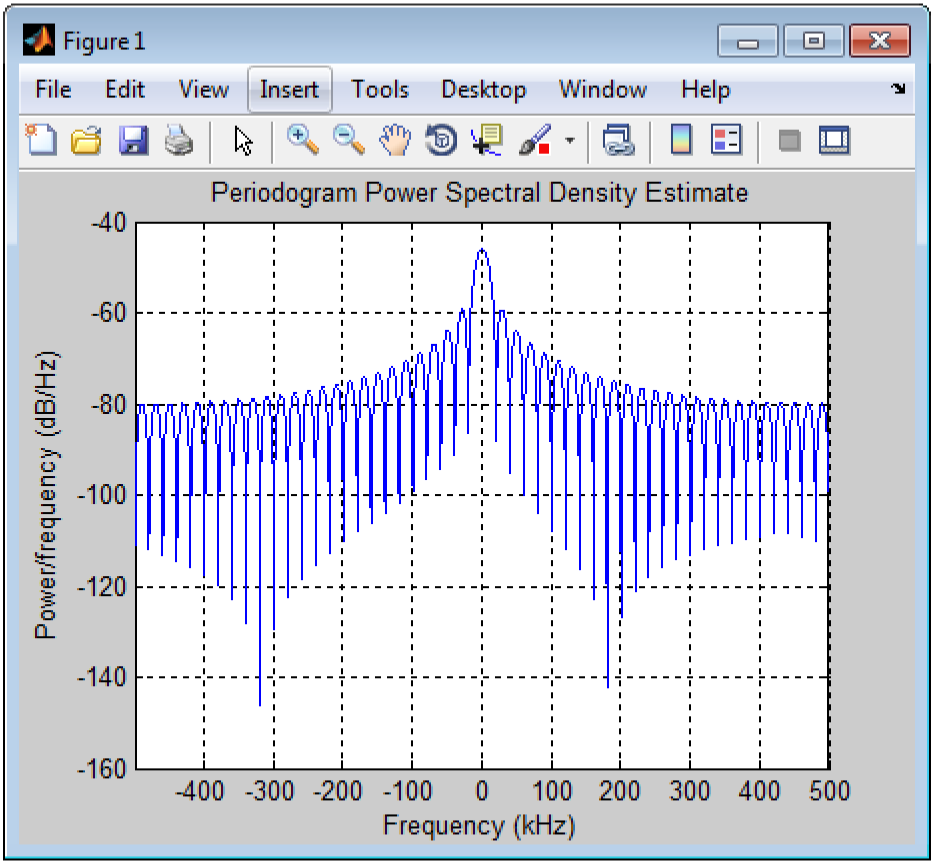


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Periodogram Power Spectral Density Estimate
-40
-60
-80
-100
-120
-140
-160
Power/frequency (dB/Hz)
-400
-300
-200
-100
0
100
200
300
400
500
Frequency (kHz)

Detect the range.

```
[~,rngidx] = findpeaks(pow2db(Pxx/max(Pxx)),...
    'MinPeakHeight',-5);
```

```
rngfreq = F(rngidx);
re = stretchfreq2rng(rngfreq,hs.SweepSlope,...
    hs.ReferenceRange,c);
```

See Also

`stretchfreq2rng`

Related Examples

- Range Estimation Using Stretch Processing

Concepts

- “Stretch Processing”

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 1-1065.
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

Sensor array

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

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

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')
```

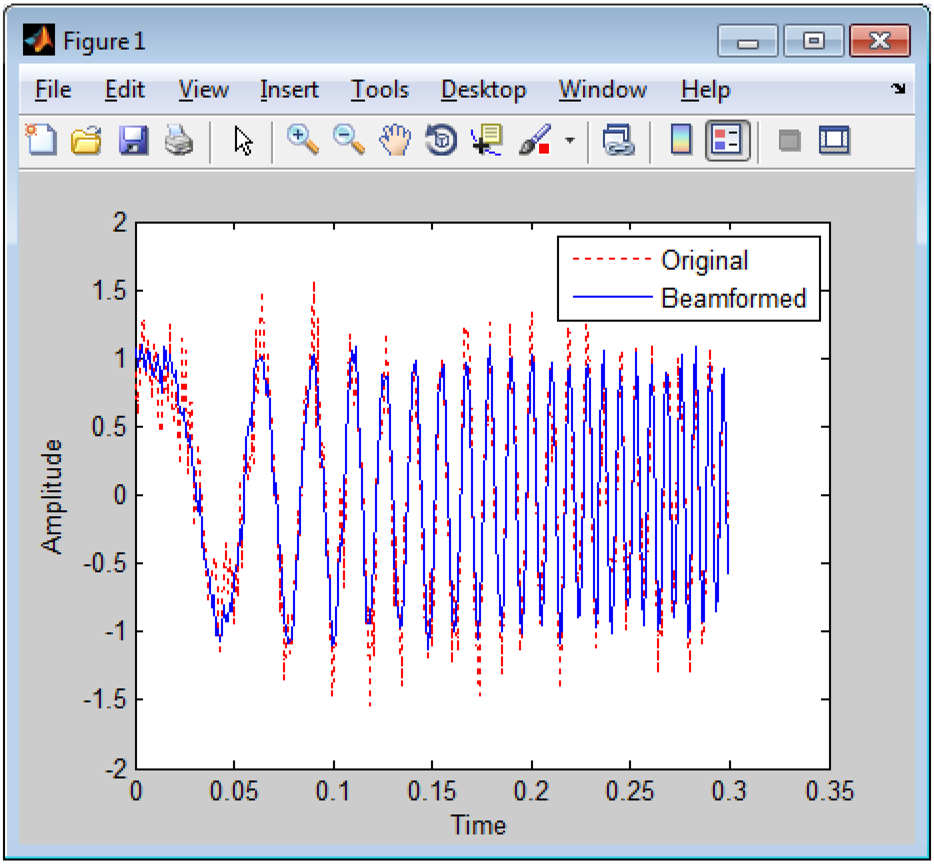


Figure 1
File Edit View Insert Tools Desktop Window Help
Original
Beamformed
Amplitude
Time
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

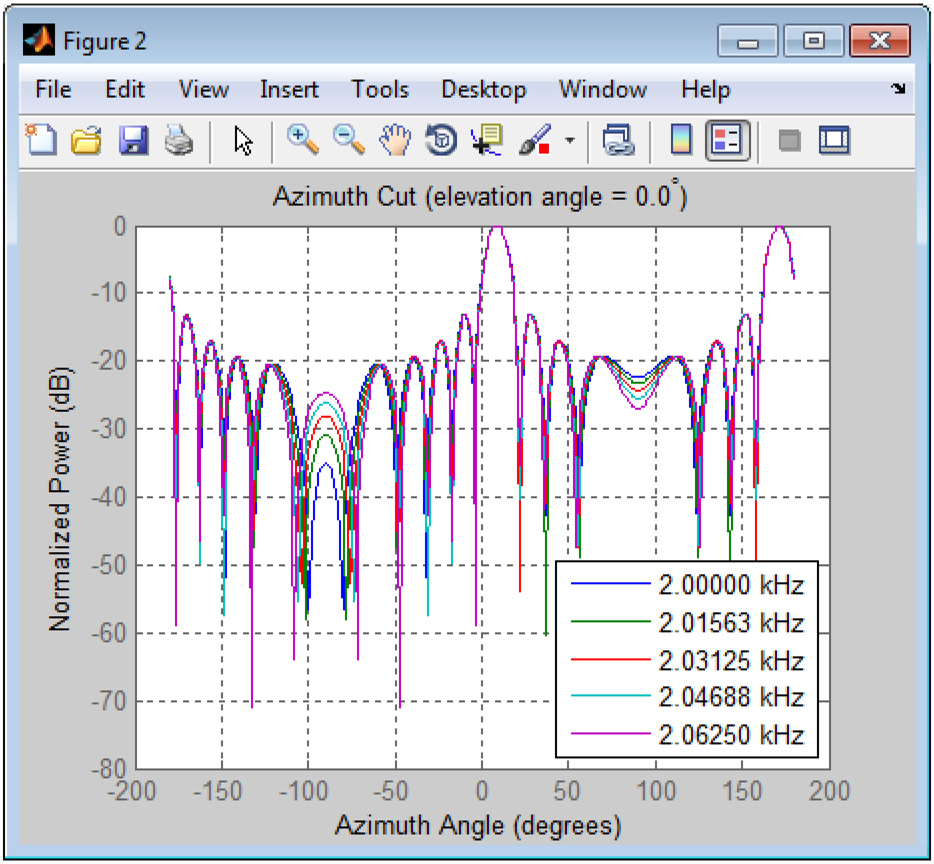


Figure 2
File
Edit
View
Insert
Tools
Desktop
Window
Help
Azimuth Cut (elevation angle = 0.0°)
Normalized Power (dB)
0
-10
-20
-30
-40
-50
-60
-70
-80
-200
-150
-100
-50
0
50
100
150
200
Azimuth Angle (degrees)
2.00000 kHz
2.01563 kHz
2.03125 kHz
2.04688 kHz
2.06250 kHz

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"

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`.

phased.SubbandPhaseShiftBeamformer.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.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.

---

phased.SubbandPhaseShiftBeamformer.step

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`

phased.SumDifferenceMonopulseTracker

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 1-1081.
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`

phased.SumDifferenceMonopulseTracker.clone

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.

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.

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 1-1091.
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.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.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.

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`

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 1-1101.
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');
```

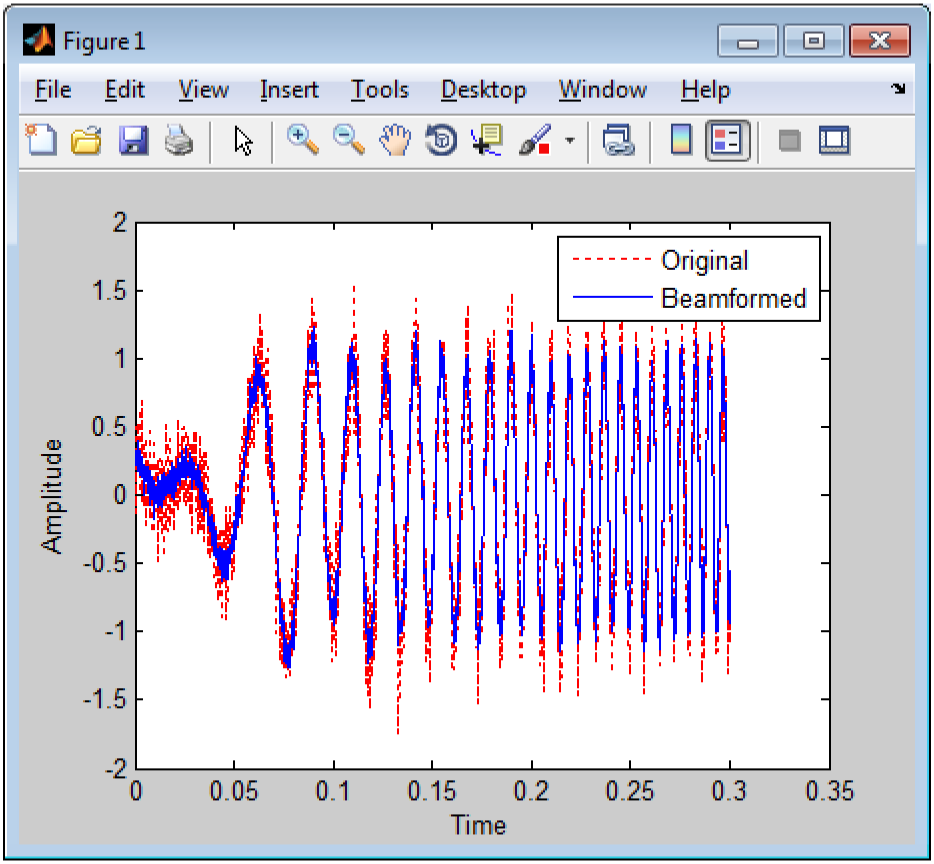


Figure 1
File Edit View Insert Tools Desktop Window Help
Original
Beamformed
Amplitude
Time
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

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.

phased.TimeDelayBeamformer.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 `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.

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.TimeDelayBeamformer.step

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 1-1114.
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');
```

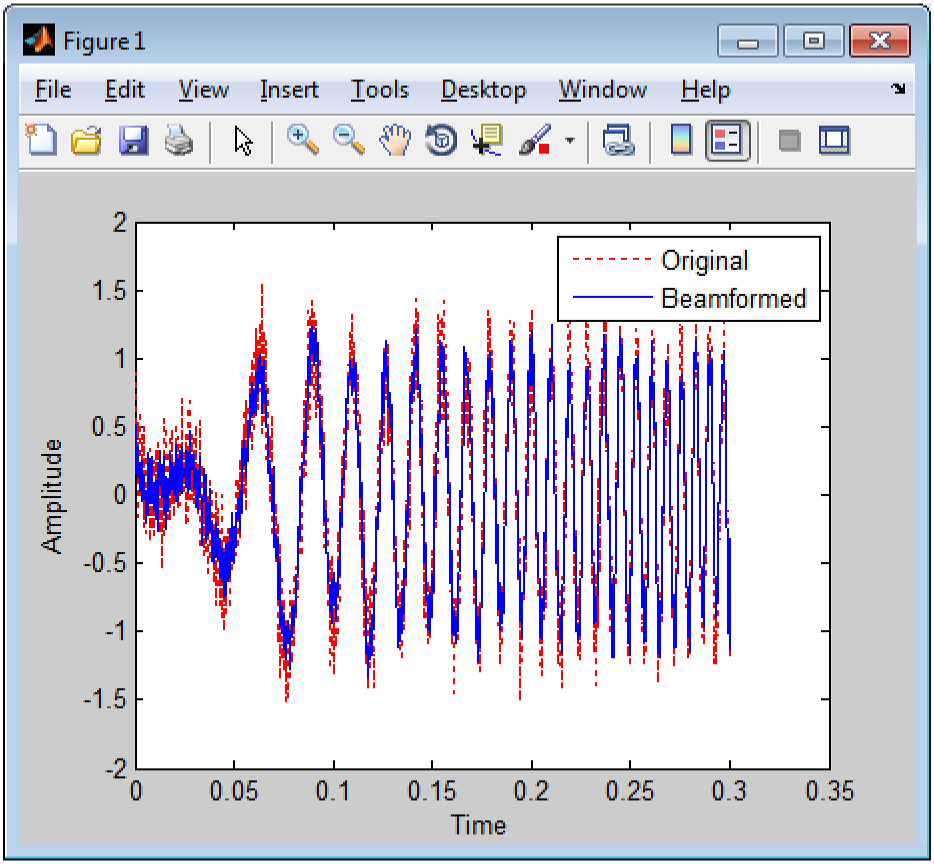


Figure 1
File Edit View Insert Tools Desktop Window Help
Original
Beamformed
Amplitude
Time
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35

Algorithms

The 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”

phased.TimeDelayLCMVBeamformer.clone

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.

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`

phased.TimeVaryingGain

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 1-1130.
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');
```

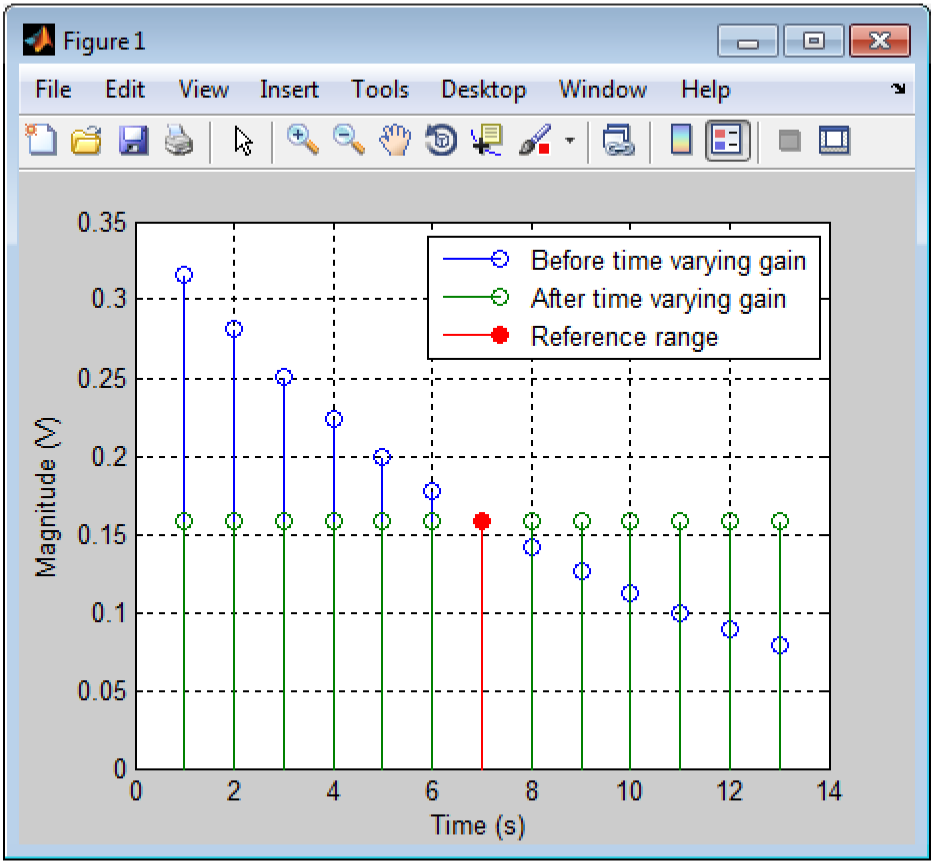


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Before time varying gain
After time varying gain
Reference range
Magnitude (V)
Time (s)
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
2
4
6
8
10
12
14

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`

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.

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.

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.TimeVaryingGain.step

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');
```

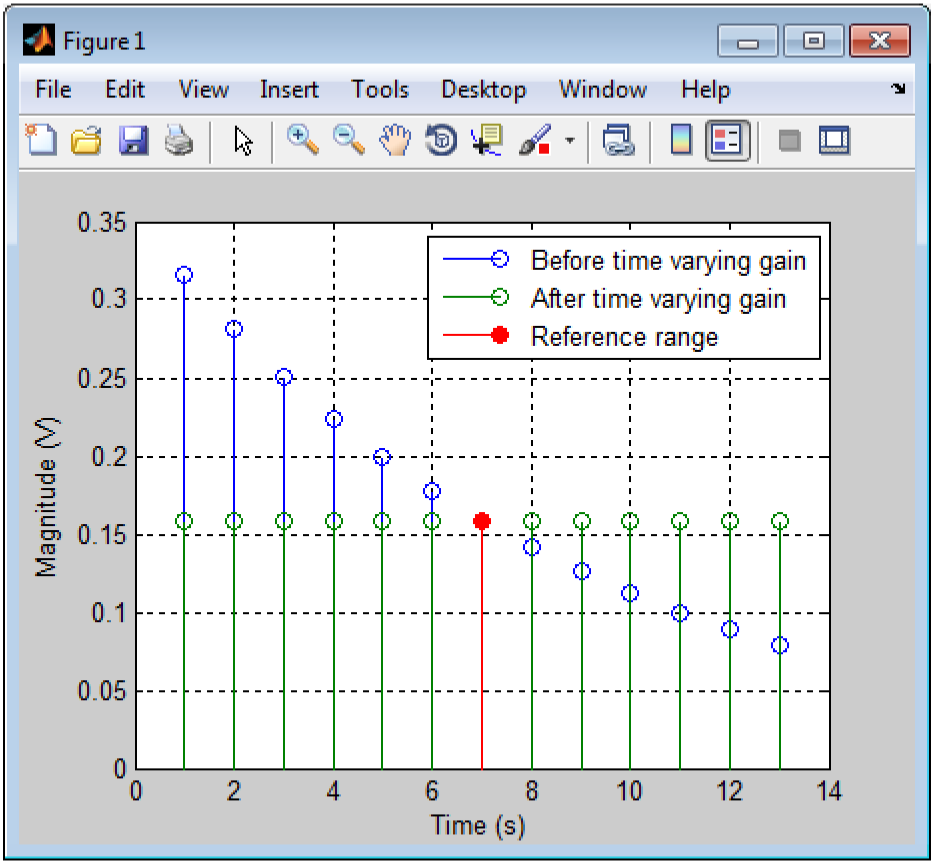


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
Magnitude (V)
0
2
4
6
8
10
12
14
Time (s)
Before time varying gain
After time varying gain
Reference range

phased.Transmitter

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 1-1140.

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`

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.

phased.Transmitter.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 `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.

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.Transmitter.reset

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'`.

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);
```

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 1-1153.
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`

Taper

Element tapering

Element tapering specified as a complex-valued scalar or a complex-valued 1-by-*N* row vector. In this vector, *N* represents the number of elements of the array. Tapers, also known as weights, are applied to each sensor elements in the sensor array and modify both the amplitude and phase of the received data. If `'Taper'` is a scalar, the same weights are applied to each element. If `'Taper'` is a vector, each weight is applied to the corresponding sensor element.

Default: `1`

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
getTaper	Array element tapers
isLocked	Locked status for input attributes and nontunable properties
isPolarizationCapable	Polarization capability
plotGratingLobeDiagram	Plot grating lobe diagram of array
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 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)
```

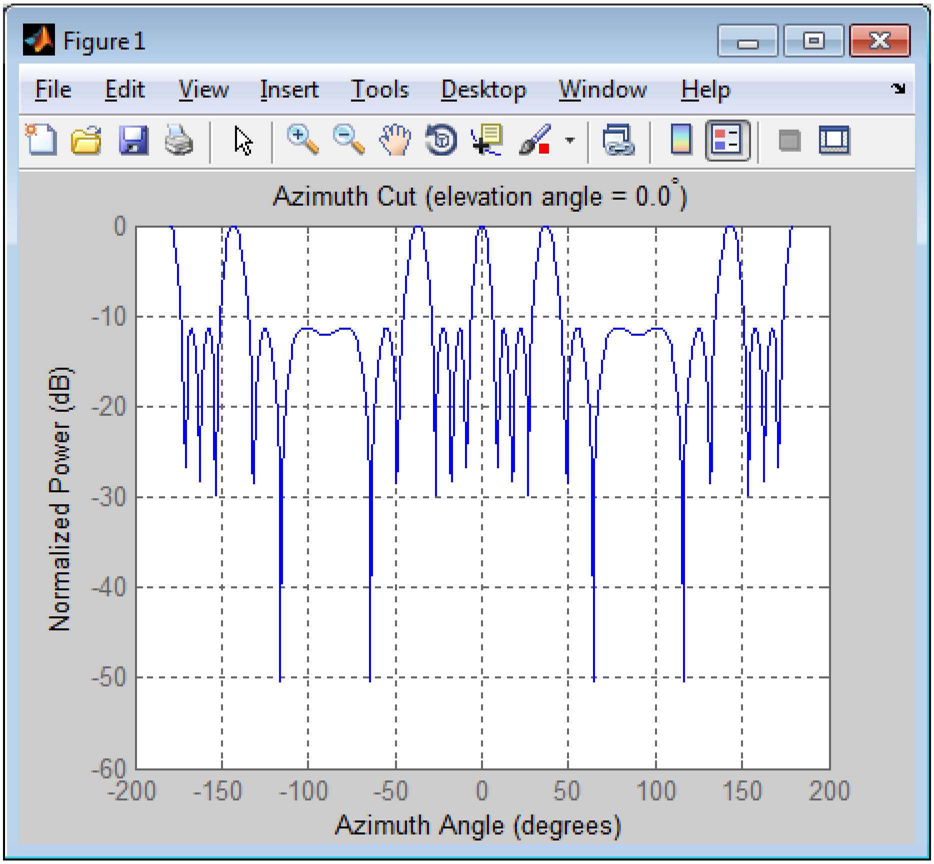


Figure 1
File Edit View Insert Tools Desktop Window Help
Azimuth Cut (elevation angle = 0.0°)
Normalized Power (dB)
0
-10
-20
-30
-40
-50
-60
-200
-150
-100
-50
0
50
100
150
200
Azimuth Angle (degrees)

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');
```

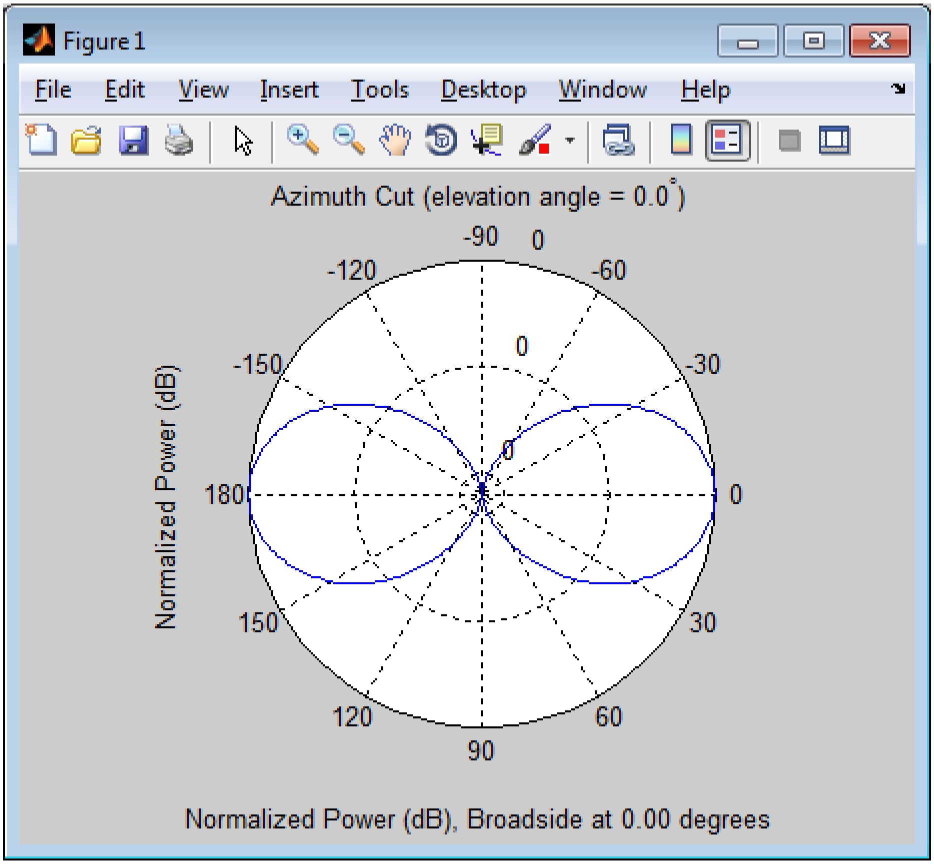


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Azimuth Cut (elevation angle = 0.0°)
-90
0
-120
-60
0
-150
-30
0
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

Response of an Array of Polarized Short-Dipole Antennas

Build a uniform line array of 5 short-dipole sensor elements. Because short dipoles support polarization, the array should as well. Verify that it supports polarization by looking at the output of `isPolarizationCapable`. Then, draw the array, showing the tapering.

Build the array and display its shape using the `viewArray` method.

```
h = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[100e6 1e9],'AxisDirection','Z');
ha = phased.ULA('NumElements',5,'Element',h,...
    'Taper',[.5,.7,1,.7,.5]);
viewArray(ha,'ShowTaper',true,'ShowIndex','All')
isPolarizationCapable(ha)
```

```
ans =

     1
```

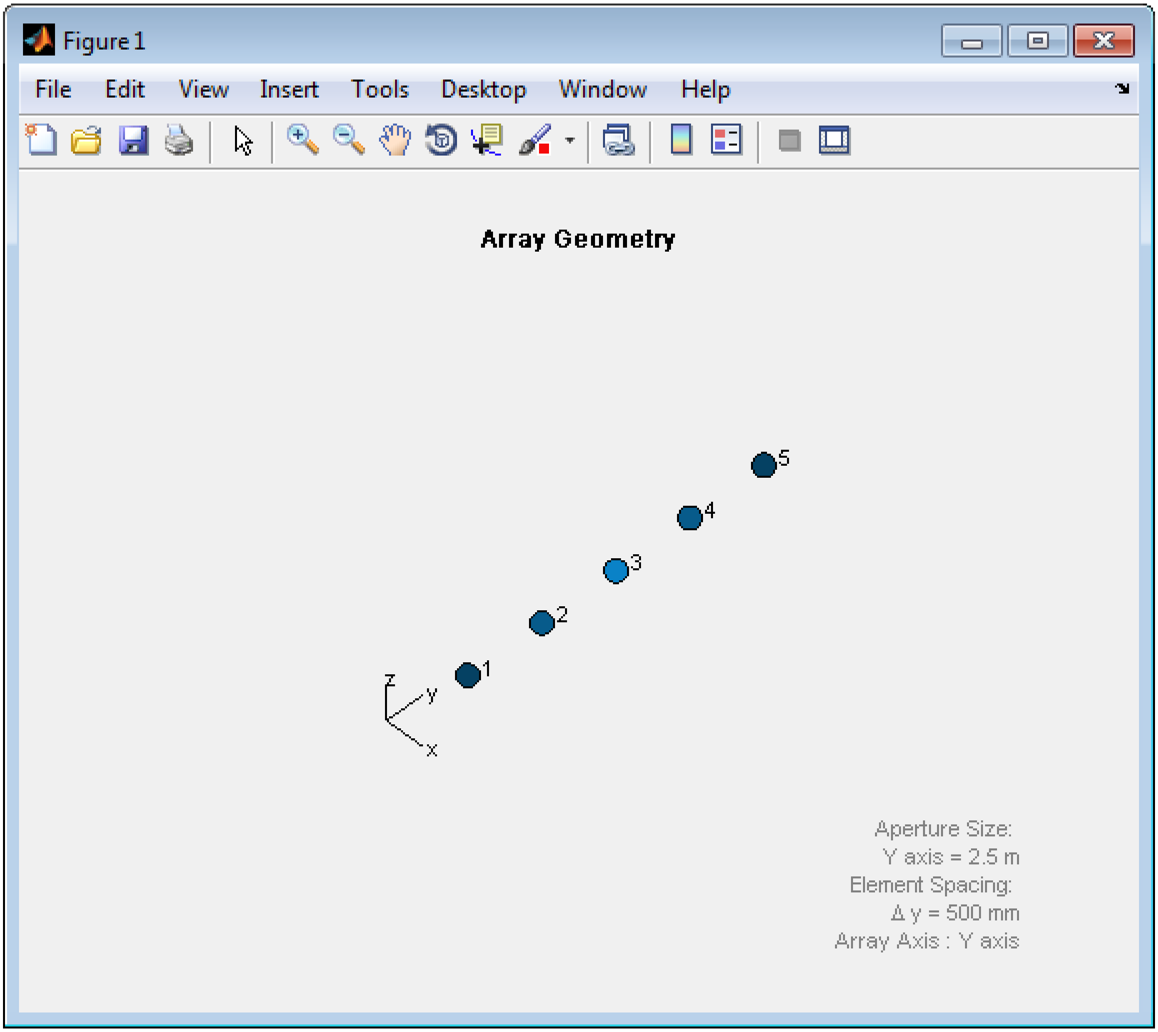


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Array Geometry
5
4
3
2
1
z
y
x
Aperture Size:
Y axis = 2.5 m
Element Spacing:
Δ y = 500 mm
Array Axis : Y axis

Display the response.

```
fc = 150e6;
ang = [10];
```

```
resp = step(ha,fc,ang);

resp =

    H: [5x2 double]
    V: [5x2 double]

resp.V

   -0.6124   -0.6124
   -0.8573   -0.8573
   -1.2247   -1.2247
   -0.8573   -0.8573
   -0.6124   -0.6124
```

Plot the vertical polarization response.

```
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','Az','Format',...
    'Polar','Polarization','V');
```

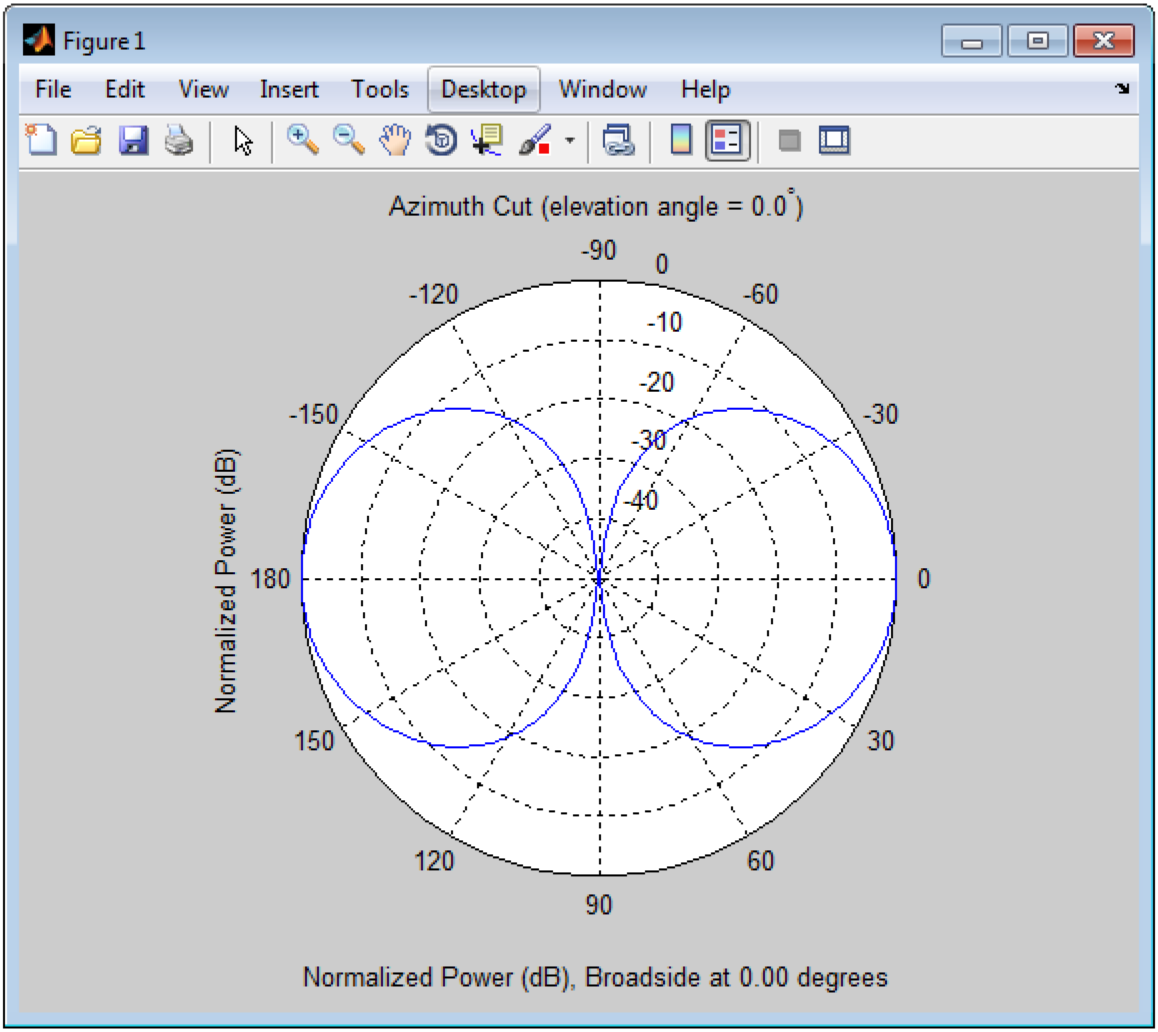


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Azimuth Cut (elevation angle = 0.0°)
-90
0
-120
-60
-10
-20
-150
-30
-30
-40
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

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.CrossedDipoleAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement``phased.ShortDipoleAntennaElement` | `phased.URA`

Related Examples

- Phased Array Gallery

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`.

phased.ULA.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 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)
POS = getElementPosition(H,ELEIDX)
```

Description `POS = getElementPosition(H)` returns the element positions of the ULA System object, `H` . `POS` is a 3-by-*N* matrix, where *N* is the number of elements in `H`. Each column of `POS` defines the position of an element in the local coordinate system, in meters, using the form `[x; y; z]`. 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.

`POS = getElementPosition(H,ELEIDX)` returns only the positions of the elements that are specified in the element index vector `ELEIDX`. This syntax can use any of the input arguments in the previous syntax.

Examples Construct a default ULA, and obtain the element positions.

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

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.

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.ULA.getTaper

Purpose

Array element tapers

Syntax

`wts = getTaper(h)`

Description

`wts = getTaper(h)` returns the tapers, `wts`, applied to each element of the phased uniform line array (ULA), `h`. Tapers are often referred to as weights.

Input Arguments

h - Uniform line array

`phased.ULA` System object

Uniform line array specified as a `phased.ULA` System object.

Output Arguments

wts - Array element tapers

N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1 complex-valued vector, where *N* is the number of elements in the array.

Examples

ULA with Taylor Window Taper

Construct a 5-element ULA with a Taylor window taper. Then, obtain the element taper values.

```
taper = taylorwin(5)';
ha = phased.ULA(5,'Taper',taper);
w = getTaper(ha)
```

```
w =

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

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.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h - Uniform line array

Uniform line array specified as a `phased.ULA` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability flag returned as a Boolean value `true` if the array supports polarization or `false` if it does not.

Examples

ULA of Short-Dipole Antenna Elements Supports Polarization

Show that an array of `phased.ShortDipoleAntennaElement` antenna elements supports polarization.

```
h = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[1e9 10e9]);
ha = phased.ULA('NumElements',3,'Element',h);
isPolarizationCapable(ha)
```

```
ans =

     1
```

The returned value `true (1)` shows that this array supports polarization.

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 specified as a scalar or 1-by-K row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the array response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'Weights'

Weight values applied to the array, specified as a length-N column vector or N-by-M matrix. The dimension N is the number of elements in the array. The interpretation of M depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-M row vector	Apply one set of weights for the same single frequency or all M frequencies.
N-by-M matrix	Scalar	Apply all of the M different columns in `Weights` for the same single frequency.
	1-by-M row vector	Apply each of the M different columns in `Weights` for the corresponding frequency in `FREQ`.

'AzimuthAngles'

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U/V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

Examples

Line Plot Showing Multiple Frequencies

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)
```

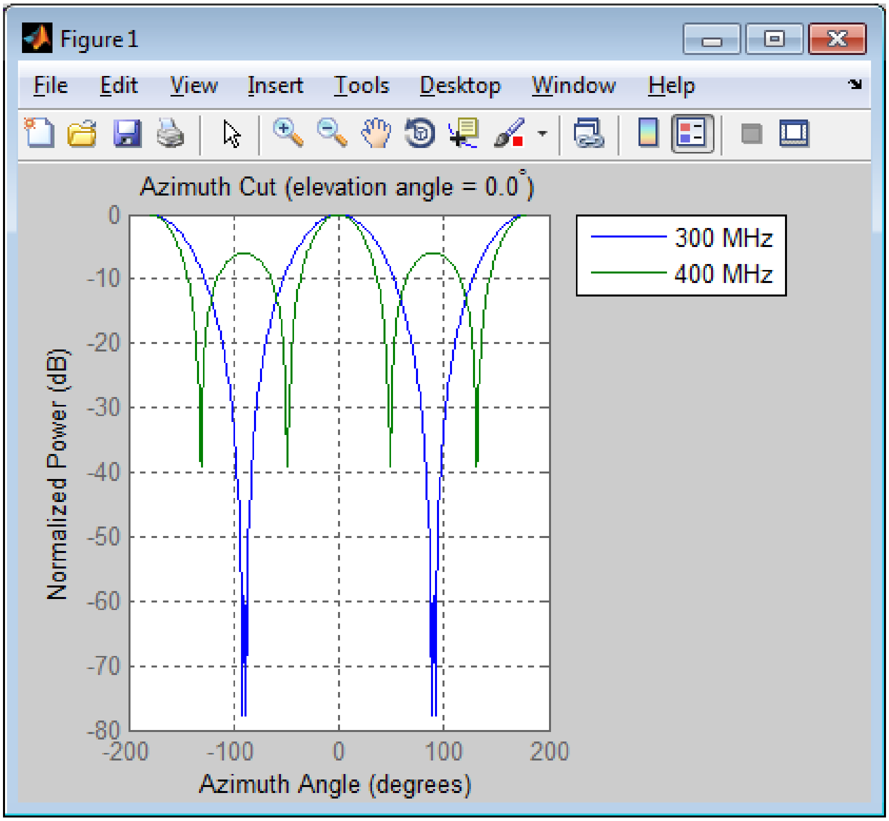


Figure 1
File Edit View Insert Tools Desktop Window Help
Azimuth Cut (elevation angle = 0.0°)
Normalized Power (dB)
0
-10
-20
-30
-40
-50
-60
-70
-80
-200
-100
0
100
200
Azimuth Angle (degrees)
300 MHz
400 MHz

Plot Azimuth Response of 4-Element ULA

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');
```

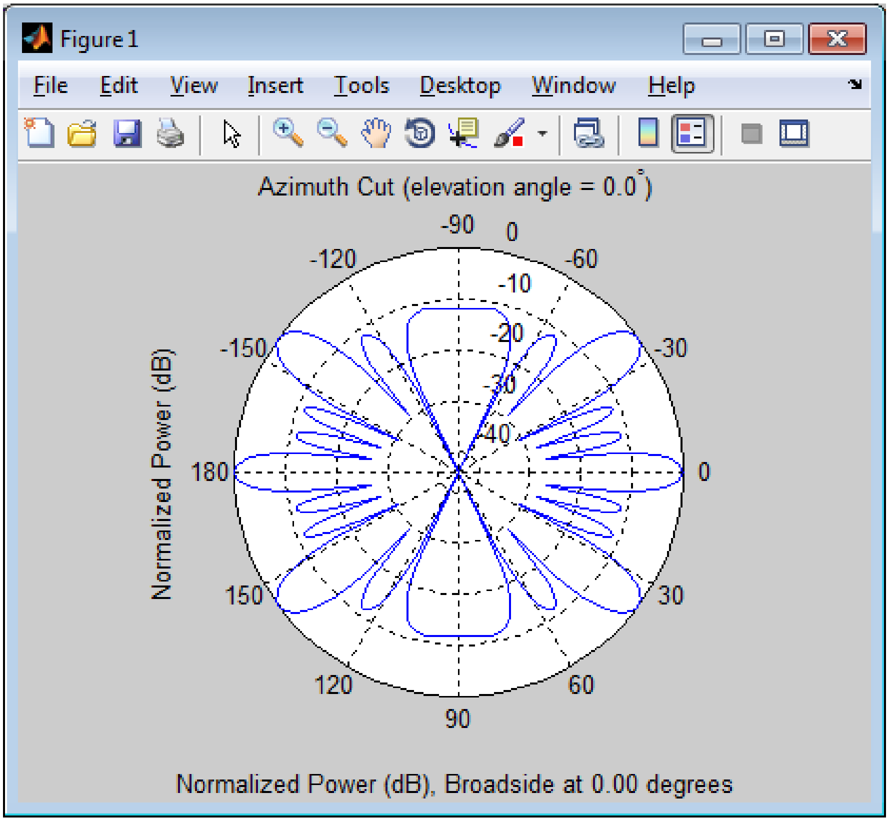


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Azimuth Cut (elevation angle = 0.0°)
-90
0
-120
-60
-10
-20
-30
-40
-150
-30
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

Azimuth Response of Weighted 11-Element ULA

Construct an 11–element ULA array of backbaffled omnidirectional microphones for beamforming the direction of arrival of sound in air. The elements are spaced four centimeters apart and have a frequency response in the band 2000–8000 Hz. Then, use `plotResponse` to display an azimuth cut of the array's response at 5000 Hz. Apply both uniform weights and Taylor window weights to the array at a single frequency using the `Weights` parameter. Using the `AzimuthAngles` parameter, limit the display to ±45° in 0.1° increments. A typical value for the speed of sound in air is 343 m/s.

```
s_omni = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[2000,8000],...
    'BackBaffled',true);
s_ula = phased.ULA(11,'Element',s_omni,...
    'ElementSpacing',0.04);
c = 343.0;
fc = 5000;
wts = taylorwin(11);
plotResponse(s_ula,fc,c,'RespCut','Az',...
    'Format','Polar',...
    'Weights',[ones(11,1),wts],...
    'AzimuthAngles',[-45:.1:45]);
```

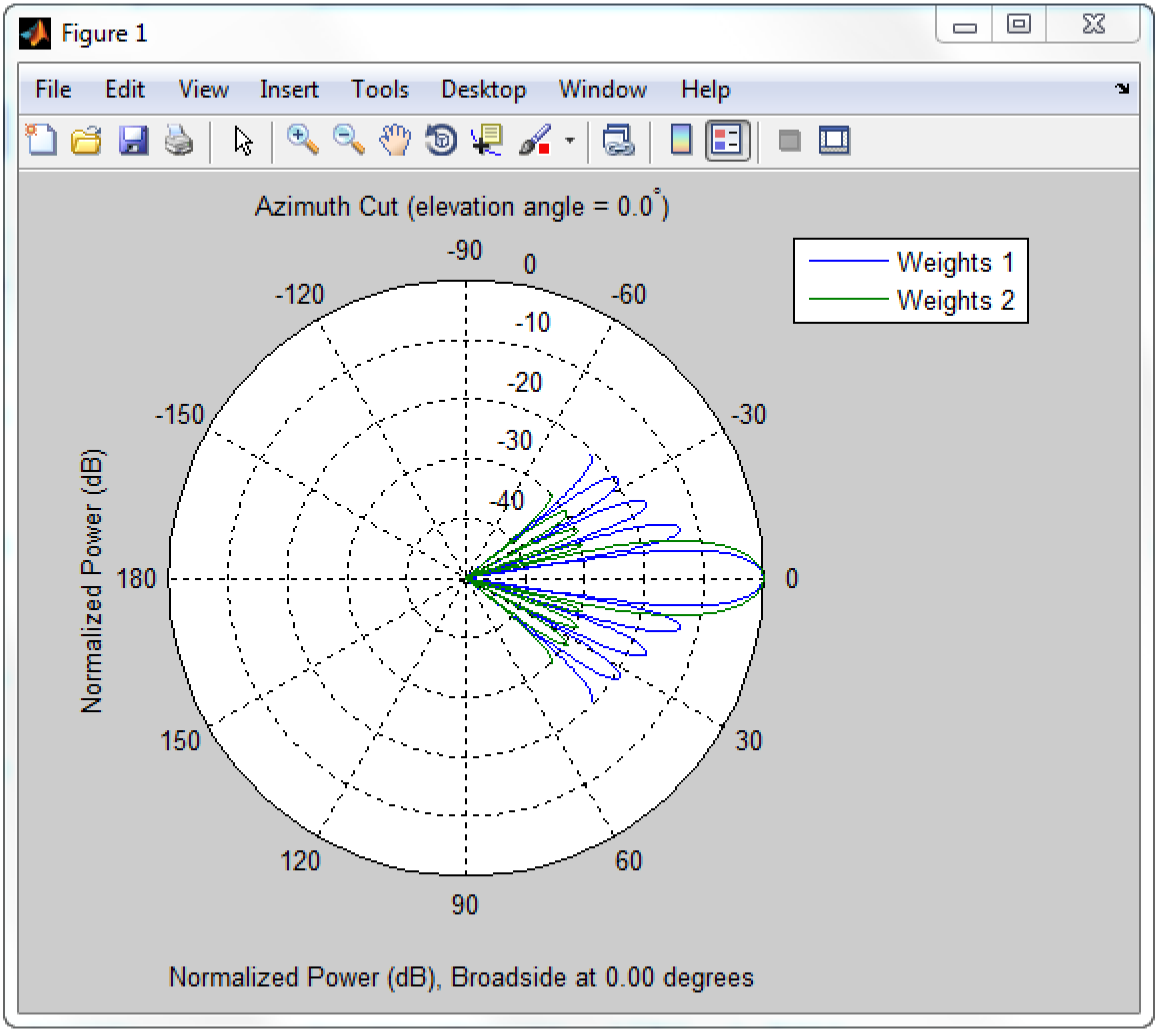


Figure 1
File Edit View Insert Tools Desktop Window Help
Azimuth Cut (elevation angle = 0.0°)
Weights 1
Weights 2
-90
0
-120
-60
-10
-20
-30
-40
-150
-30
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

The plot shows that the tapered set of weights reduces the adjacent sidelobes while broadening the main lobe.

Plot Directivity of 11-Element ULA of Cosine-Pattern Antennas

Construct an 11-element ULA of cosine antenna elements spaced one-half wavelength apart. Then, plot an azimuth cut of its directivity. Assume the operating frequency is 1.5 GHz and the wave propagation speed is the speed of light.

```
fc = 1.5e9;
c = physconst('Lightspeed');
lambda = c/fc;
sCos = phased.CosineAntennaElement('FrequencyRange',...
    [1e9 2e9],'CosinePower',[2.5,3.5]);
sULA = phased.ULA(11,0.5*lambda,'Element',sCos);
plotResponse(sULA,fc,c,'RespCut','Az','Unit','dbi');
```

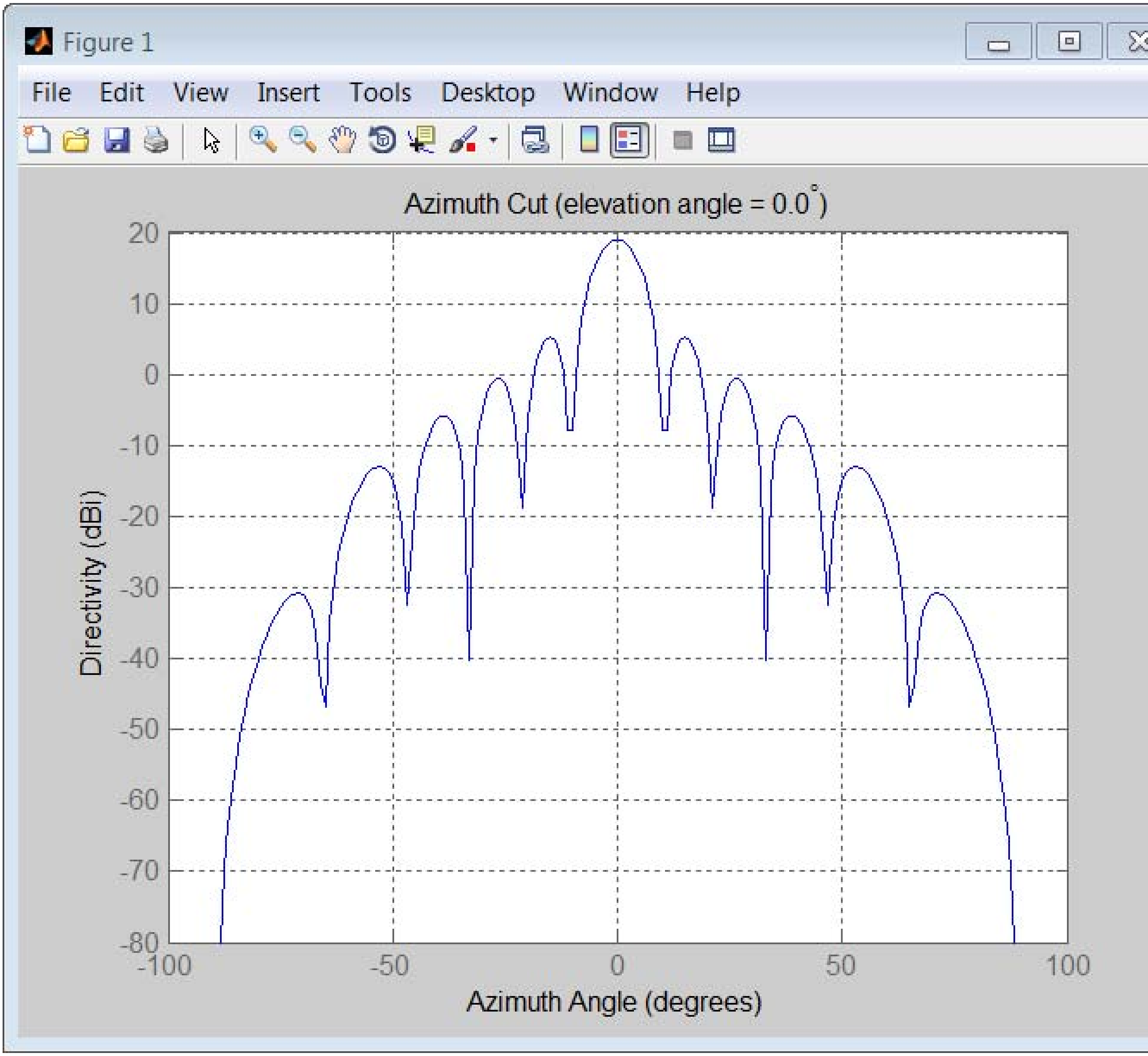


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Azimuth Cut (elevation angle = 0.0°)
Directivity (dBi)
20
10
0
-10
-20
-30
-40
-50
-60
-70
-80
-100
-50
0
50
100
Azimuth Angle (degrees)

The directivity attains its maximum value, 18 dB, at 0° azimuth.

See Also

`uv2azel` | `azel2uv`

phased.ULA.plotGratingLobeDiagram

Purpose

Plot grating lobe diagram of array

Syntax

```
plotGratingLobeDiagram(H,FREQ)
plotGratingLobeDiagram(H,FREQ,ANGLE)
plotGratingLobeDiagram(H,FREQ,ANGLE,C)
plotGratingLobeDiagram(H,FREQ,ANGLE,C,F0)
hPlot = plotGratingLobeDiagram( ___ )
```

Description

`plotGratingLobeDiagram(H,FREQ)` plots the grating lobe diagram of an array in the *u-v* coordinate system. The System object `H` specifies the array. The argument `FREQ` specifies the signal frequency and phase-shifter frequency. The array, by default, is steered to 0° azimuth and 0° elevation.

A grating lobe diagram displays the positions of the peaks of the narrowband *array pattern*. The array pattern depends only upon the geometry of the array and not upon the types of elements which make up the array. Visible and nonvisible grating lobes are displayed as open circles. Only grating lobe peaks near the location of the mainlobe are shown. The mainlobe itself is displayed as a filled circle.

`plotGratingLobeDiagram(H,FREQ,ANGLE)`, in addition, specifies the array steering angle, `ANGLE`.

`plotGratingLobeDiagram(H,FREQ,ANGLE,C)`, in addition, specifies the propagation speed by `C`.

`plotGratingLobeDiagram(H,FREQ,ANGLE,C,F0)`, in addition, specifies an array phase-shifter frequency, `F0`, that differs from the signal frequency, `FREQ`. This argument is useful when the signal no longer satisfies the narrowband assumption and, allows you to estimate the size of beam squint.

`hPlot = plotGratingLobeDiagram( ___ )` returns the handle to the plot for any of the input syntax forms.

Input Arguments

H

Antenna or microphone array, specified as a System object.

FREQ

Signal frequency, specified as a scalar. Frequency units are hertz. Values must lie within a range specified by the frequency property of the array elements contained in `H.Element`. The frequency property is named `FrequencyRange` or `FrequencyVector`, depending on the element type.

ANGLE

Array steering angle, specified as either a 2-by-1 vector or a scalar. If `ANGLE` is a vector, it takes the form `[azimuth;elevation]`. The azimuth angle must lie in the range `[-180 ,180 ]`. The elevation angle must lie in the range `[-90 ,90 ]`. All angle values are specified in degrees. If the argument `ANGLE` is a scalar, it specifies only the azimuth angle where the corresponding elevation angle is 0°.

Default: `[0;0]`

C

Signal propagation speed, specified as a scalar. Units are meters per second.

Default: Speed of light in vacuum

F0

Phase-shifter frequency of the array, specified as a scalar. Frequency units are hertz When this argument is omitted, the phase-shifter frequency is assumed to be the signal frequency, `FREQ`.

Default: `FREQ`

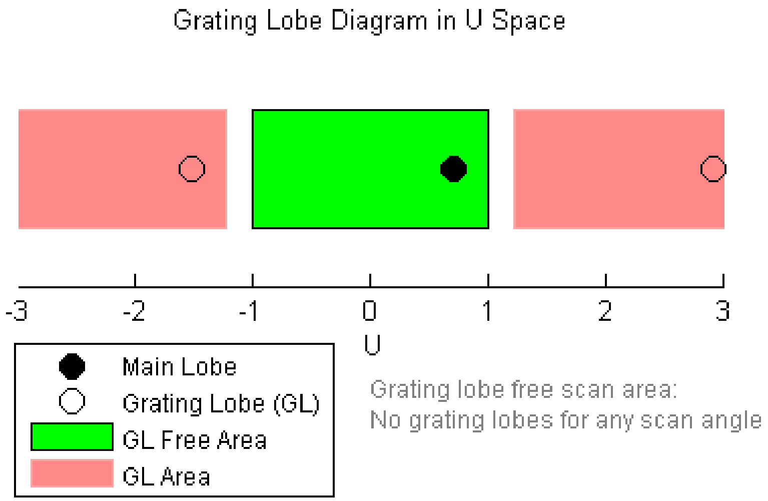
Examples

Create Grating Lobe Diagram for ULA

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing less than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 3 GHz and the spacing between elements is 0.45 of the wavelength. All elements are isotropic antenna elements. Steer the array in the direction 45 degrees in azimuth and 0 degrees in elevation.

```
c = physconst('LightSpeed');
f = 3e9;
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,...
    'ElementSpacing',0.45*lambda);
plotGratingLobeDiagram(sULA,f,[45;0],c);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region is defined by the direction cosine limits between [-1,1] and is marked by the two vertical black lines. Because the array spacing is less than one-half wavelength, there are no grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it coincides with the visible region.

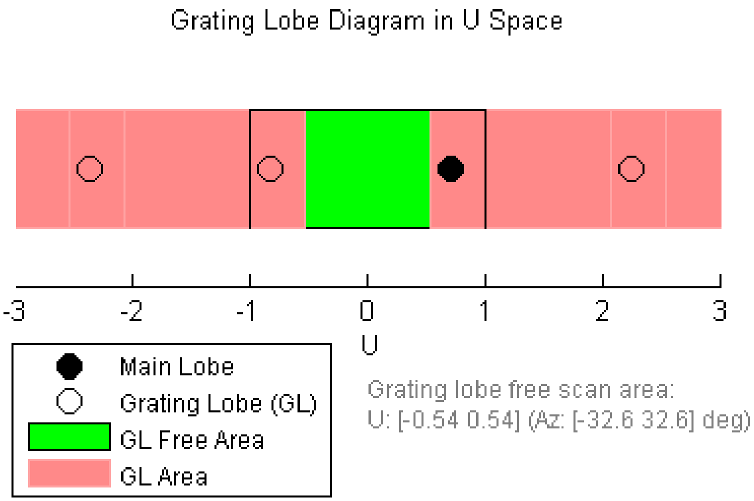
The white area of the diagram indicates a region where no grating lobes are possible.

Create Grating Lobe Diagram for Undersampled ULA

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing greater than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 3 GHz and the spacing between elements is 0.65 of a wavelength. All elements are isotropic antenna elements. Steer the array in the direction 45 degrees in azimuth and 0 degrees in elevation.

```
c = physconst('LightSpeed');
f = 3e9;
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,'ElementSpacing',0.65*la
plotGratingLobeDiagram(sULA,f,[45;0],c);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region, marked by the two black vertical lines, corresponds to arrival angles between -90 and 90 degrees. The visible region is defined by the direction cosine limits $-1 \le u \le 1$. Because the array spacing is greater than one-half wavelength, there is now a grating lobe in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those for which $-3 \le u \le 3$ are shown.

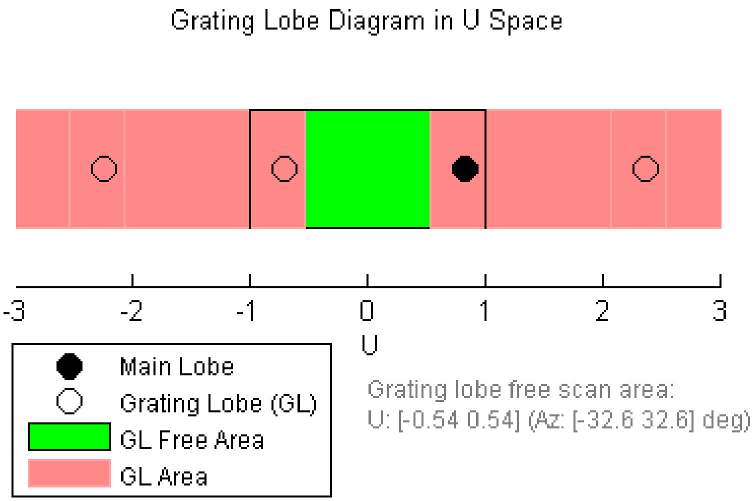
The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region.

Create Grating Lobe Diagram for ULA With Different Phase-Shifter Frequency

Plot the grating lobe diagram for a 4-element uniform linear array having element spacing greater than one-half wavelength. Apply a phase-shifter frequency that differs from the signal frequency. Grating lobes are plotted in u-v coordinates.

Assume the signal frequency is 3 GHz and the spacing between elements is 0.65 λ . All elements are isotropic antenna elements. The phase-shifter frequency is set to 3.5 GHz. Steer the array in the direction $45°$ azimuth, $0°$ elevation.

```
c = physconst('LightSpeed');
f = 3e9;
f0 = 3.5e9;
lambda = c/f;
sIso = phased.IsotropicAntennaElement;
sULA = phased.ULA('Element',sIso,'NumElements',4,...
    'ElementSpacing',0.65*lambda );
plotGratingLobeDiagram(sULA,f,[45;0],c,f0);
```



As a result of adding the shifted frequency, the mainlobe shifts right towards larger u values. The beam no longer points toward the actual source arrival angle.

The mainlobe of the array is indicated by a filled black circle. The grating lobes in the visible and nonvisible regions are indicated by empty black circles. The visible region, marked by the two black vertical lines, corresponds to arrival angles between $-90°$ and $90°$. The visible region is defined by the direction cosine limits $-1 \le u \le 1$. Because the array spacing is greater than one-half wavelength, there is now a grating lobe in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those for which $-3 \leq u \leq 3$ are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region.

Concepts

Grating Lobes

Spatial undersampling of a wavefield by an array gives rise to visible grating lobes. If you think of the wavenumber, k, as analogous to angular frequency, then you must sample the signal at spatial intervals smaller than π/k_{max} (or $\lambda_{min}/2$) in order to remove aliasing. The appearance of visible grating lobes is also known as spatial aliasing. The variable k_{max} is the largest wavenumber value present in the signal.

The directions of maximum spatial response of a ULA are determined by the peaks of the array's *array pattern* (alternatively called the *beam pattern* or *array factor*). Peaks other than the mainlobe peak are called grating lobes. For a ULA, the array pattern depends only on the wavenumber component of the wavefield along the array axis (the y-direction for the `phased.ULA` System object). The wavenumber component is related to the look-direction of an arriving wavefield by $k_y = -2\pi \sin \varphi/\lambda$. The angle φ is the broadside angle—the angle that the look-direction makes with a plane perpendicular to the array. The look-direction points away from the array to the wavefield source.

The array pattern possesses an infinite number of periodically-spaced peaks that are equal in strength to the mainlobe peak. If you steer the array to the φ_0 direction, the array pattern for a ULA has its mainlobe peak at the wavenumber value of $k_{y0} = -2\pi \sin \varphi_0/\lambda$. The array pattern has strong grating lobe peaks at $k_{ym} = k_{y0} + 2\pi\, m/d$, for any integer value m. Expressed in terms of direction cosines, the grating lobes occur at $u_m = u_0 + m\lambda/d$, where $u_0 = \sin \varphi_0$. The direction cosine, u_0, is the cosine of the angle that the look-direction makes with the y-axis and is equal to $\sin \varphi_0$ when expressed in terms of the look-direction.

In order to correspond to a physical look-direction, u_m must satisfy, $-1 \leq u_m \leq 1$. You can compute a physical look-direction angle φ_m from $\sin \varphi_m = u_m$ as long as $-1 \leq u_m \leq 1$. The spacing of grating lobes depends upon λ/d. When λ/d is small enough, multiple grating lobe peaks can correspond to physical look-directions.

The presence or absence of visible grating lobes for the ULA is summarized in this table.

Element Spacing	**Grating Lobes**
$\lambda/d \geq 2$	No visible grating lobes for any mainlobe direction.
$1 \leq \lambda/d < 2$	Visible grating lobes can exist for some range of mainlobe directions.
$\lambda/d < 1$	Visible grating lobes exist for every mainlobe direction.

References

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

See Also

`uv2azel` **|** `azel2uv`

phased.ULA.release

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.

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`

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

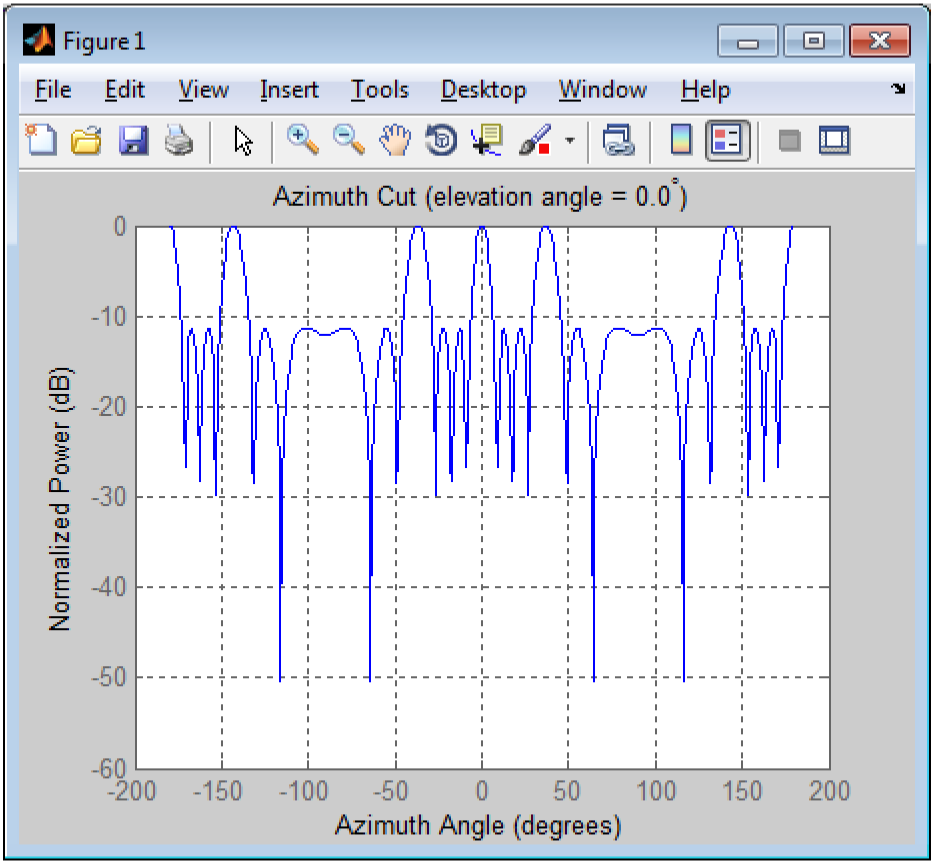
- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions N-by-M-by-L. N is the number of elements in the array. The dimension M is the number of angles specified in `ANG`. L is the number of frequencies specified in `FREQ`. For any element, the columns of `RESP` contain the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.
- If the array is capable of supporting polarization, the voltage response, `RESP`, is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions N-by-M-by-L. N is the number of elements in the array, and M is the number of angles specified in `ANG`. L is the number of frequencies specified in `FREQ`. Each column of `RESP` contains the responses of the array elements for the corresponding direction specified in `ANG`. Each of the L pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.

Examples

Response of Antenna 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');
```

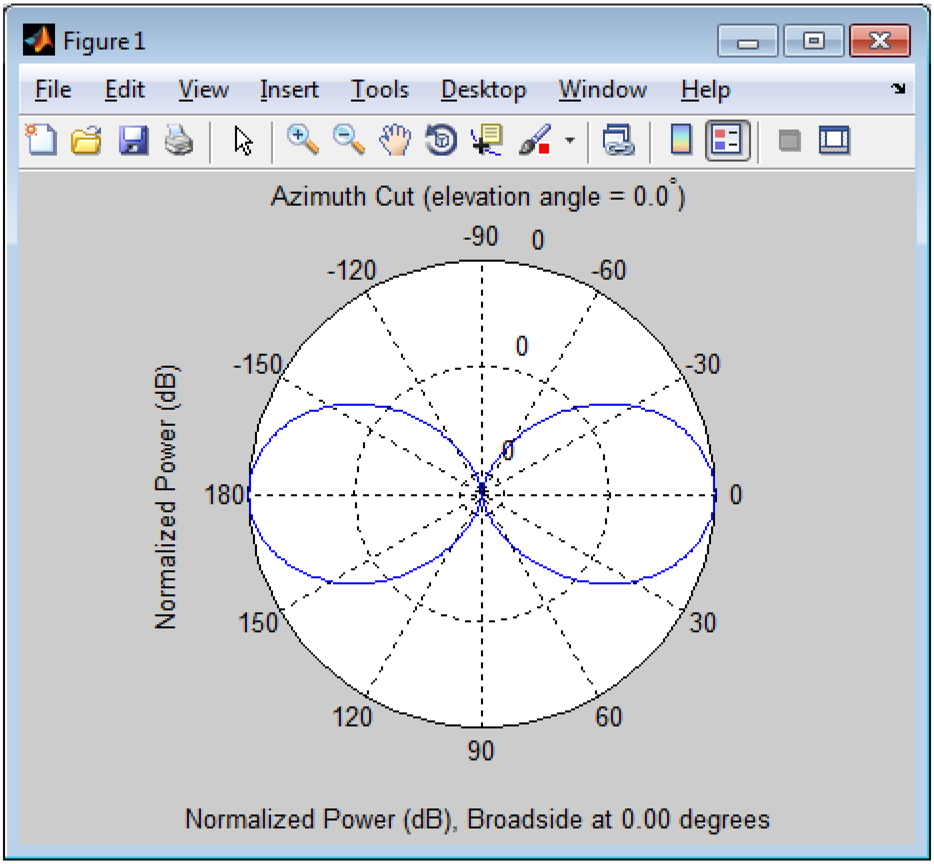


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Azimuth Cut (elevation angle = 0.0°)
-90
0
-120
-60
0
-150
-30
0
180
0
150
30
120
60
90
Normalized Power (dB)
Normalized Power (dB), Broadside at 0.00 degrees

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. `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`

'ShowTaper'

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

'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

Draw a 6-element ULA. Use the `ShowIndex` property to show the indices for the first and third elements.

```
ha = phased.ULA(6);
viewArray(ha,'ShowIndex',[1 3]);
```

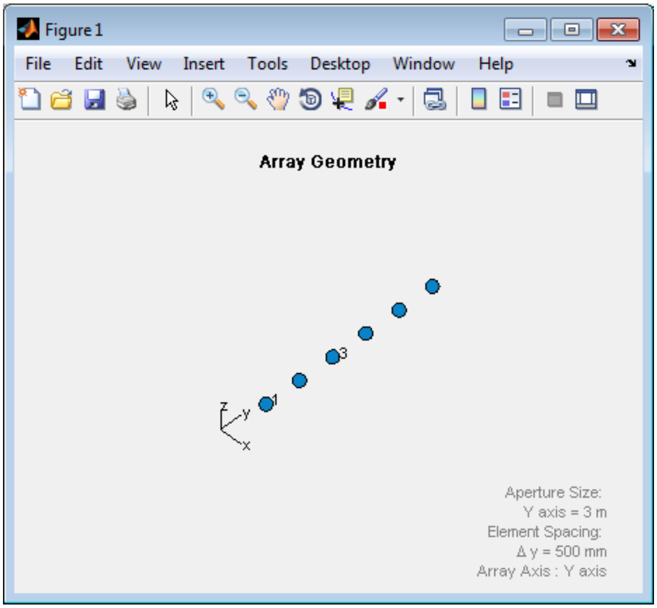


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Array Geometry
1
3
z
y
x

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 (URA).

To compute the response for each element in the array for specified directions:

1. Define and set up your uniform rectangular array. See "Construction" on page 1-1205.
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 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 (boresight) is along the positive *x*-axis.

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

`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

Phased array toolbox system object

Element specified as a Phased Array System Toolbox object. This object can be an antenna or microphone element.

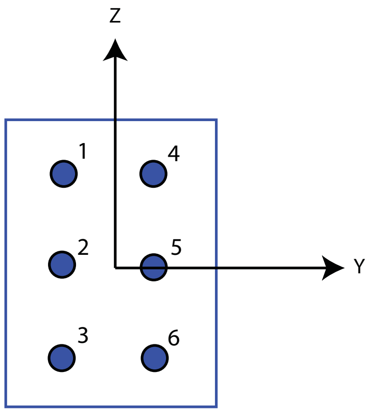
Default: An isotropic antenna element that operates between 300 MHz and 1 GHz

Size

Size of array

A 1-by-2 integer vector or a single integer containing the size of the array. If `Size` is a 1-by-2 vector, the vector has the form `[NumberOfRows, NumberOfColumns]`. If `Size` is a scalar, the array has the same number of elements in each row and column. For a URA, array elements are indexed from top to bottom along a column and continuing to the next columns from left to right. In this illustration, a `'Size'` value of `[3,2]` array has three rows and two columns.

Size and Element Indexing Order
for Uniform Rectangular Arrays
Example: Size = [3,2]



Default: `[2 2]`

ElementSpacing

Element spacing

A 1-by-2 vector or a scalar containing the element spacing of the array, expressed in meters. If `ElementSpacing` is a 1-by-2 vector, it is in the form of `[SpacingBetweenRows,SpacingBetweenColumns]`. See "Spacing Between Columns" on page 1-1208 and "Spacing Between Rows" on page 1-1209. If `ElementSpacing` is a scalar, both spacings are the same.

Default: `[0.5 0.5]`

Lattice

Element lattice

Specify the element lattice as one of `'Rectangular' | 'Triangular'`. When you set the Lattice property to `'Rectangular'`, all elements in the 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'`

Taper

Element taper

Element taper specified as a scalar or M-by-N complex-valued matrix. Tapers are applied to each element in the sensor array. Tapers are often referred to as *element weights*. M is the number of elements along the z-axis, and N is the number of elements along y-axis. M and N correspond to the values of `[NumberofRows, NumberOfColumns]` in the `Size` property. If `Taper` is a scalar, identical weights are applied to each element.

If the value of `Taper` is a matrix, a taper value is applied to the corresponding element.

Default: 1

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
getTaper	Array element tapers
isLocked	Locked status for input attributes and nontunable properties
isPolarizationCapable	Polarization capability
plotGratingLobeDiagram	Plot grating lobe diagram of array
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 Between Columns

The spacing between columns is the distance between adjacent elements in the same row.

Spacing Between Rows

The spacing between rows is the distance along the column axis direction between adjacent rows.

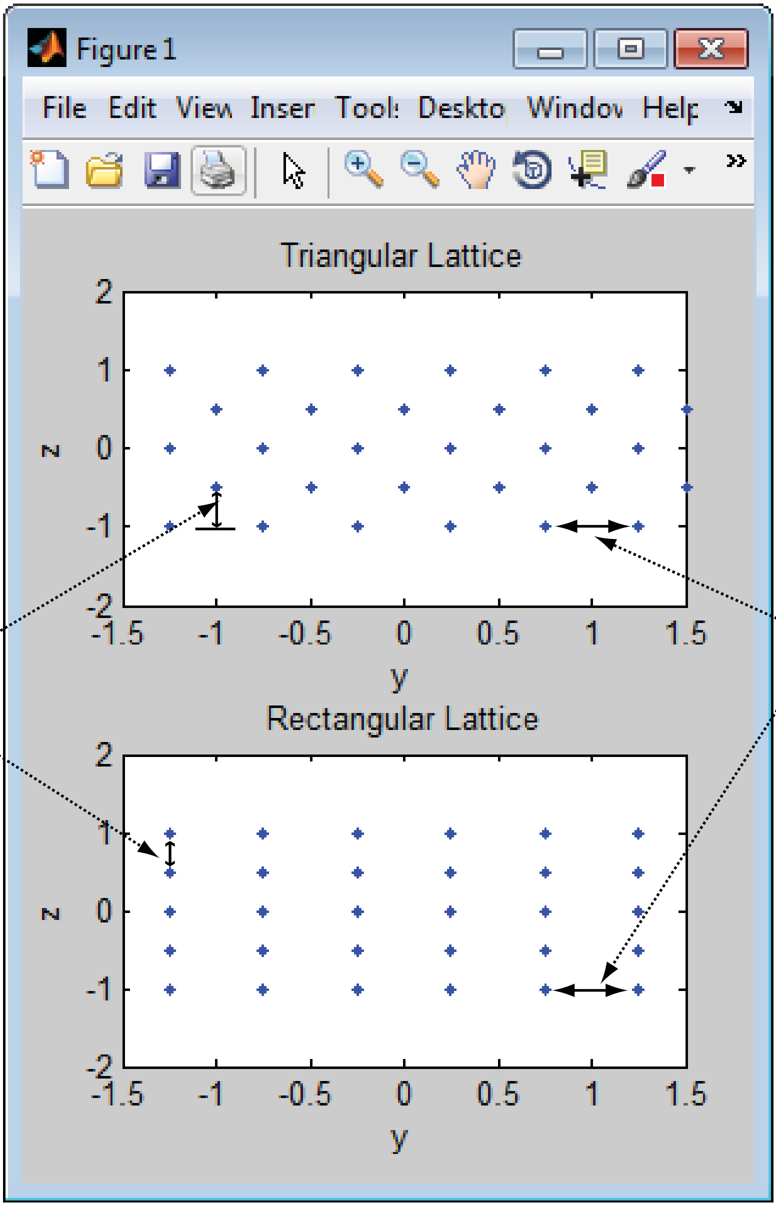


Figure 1
File Edit View Inser Tool: Deskto Windov Help
Triangular Lattice
2
1
0
-1
-2
z
-1.5
-1
-0.5
0
0.5
1
1.5
y
Rectangular Lattice
2
1
0
-1
-2
z
-1.5
-1
-0.5
0
0.5
1
1.5
y

Spacing Between
Rows

Sp

Examples

Azimuth Response of a 3-by-2 URA at Boresight

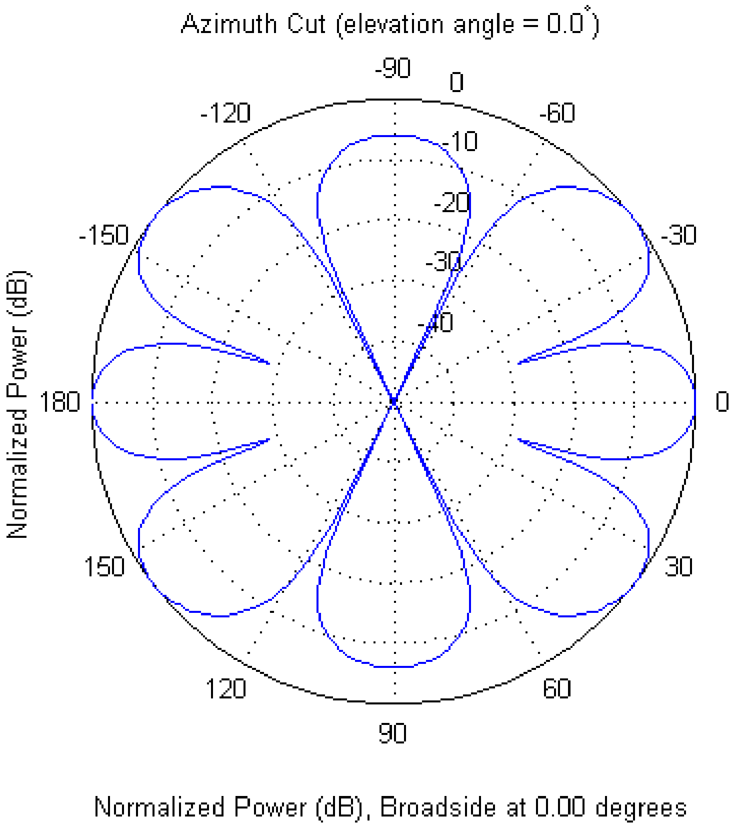
This example shows how to construct a 3-by-2 rectangular lattice URA. Find the response of each element at boresight. Assume the operating frequency is 1 GHz.

```
ha = phased.URA('Size',[3 2]);
fc = 1e9; ang = [0;0];
resp = step(ha,fc,ang);
disp(resp)
```

```
     1
     1
     1
     1
     1
     1
```

Plot the azimuth response of the array.

```
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



Compare Triangular and Rectangular Lattice URA's

This example shows how to find and plot the positions of the elements of a 5-row-by-6-column URA with a triangular lattice and a URA with a rectangular lattice. The element spacing is 0.5 meters for both lattices.

Create the arrays.

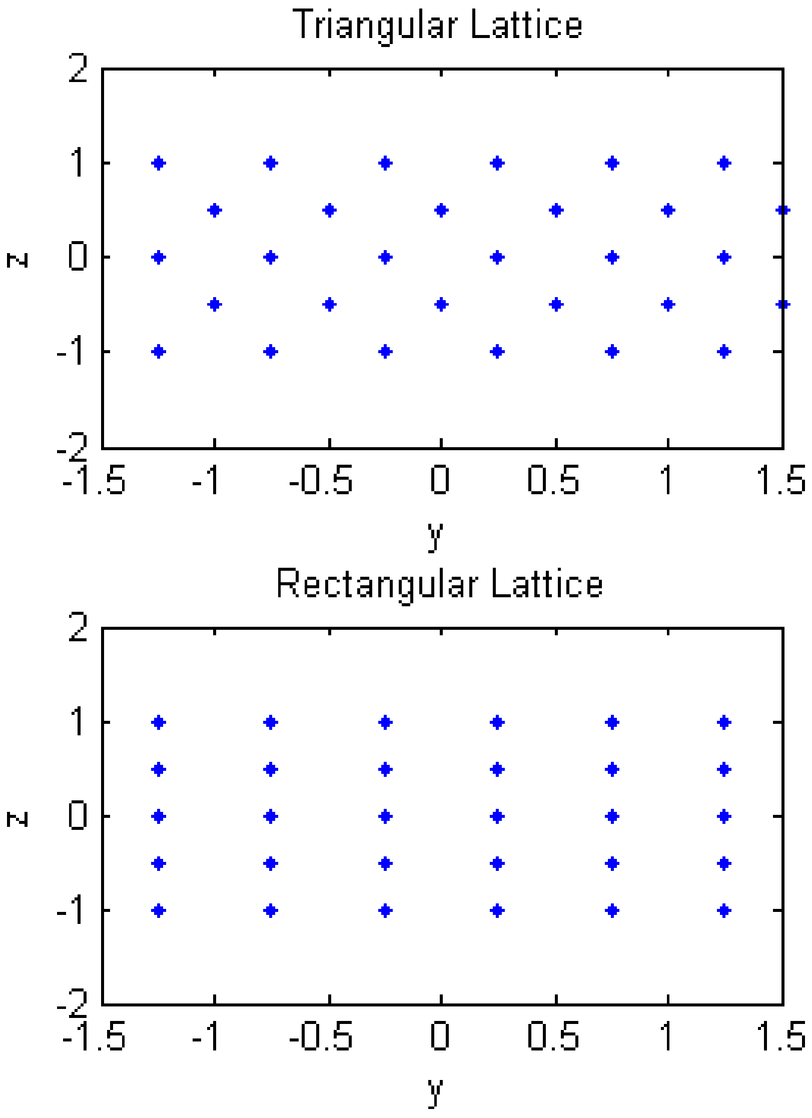
```
h_tri = phased.URA('Size',[5 6],'Lattice','Triangular');
h_rec = phased.URA('Size',[5 6],'Lattice','Rectangular');
```

Get the element y,z positions for each array. All the x coordinates are zero.

```
pos_tri = getElementPosition(h_tri);
pos_rec = getElementPosition(h_rec);
pos_yz_tri = pos_tri(2:3,:);
pos_yz_rec = pos_rec(2:3,:);
```

Plot the element positions in the yz-plane.

```
figure;
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')
```


Triangular Lattice
2
1
0
-1
-2
z
-1.5
-1
-0.5
0
0.5
1
1.5
y
Rectangular Lattice
2
1
0
-1
-2
z
-1.5
-1
-0.5
0
0.5
1
1.5
y

Adding Tapers to an Array

Construct a 5-by-2 element URA with a Taylor window taper along each column. The tapers form a 5-by-2 matrix.

```
taper = taylorwin(5);
ha = phased.URA([5,2],'Taper',[taper,taper]);
w = getTaper(ha)
```

```
w =

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

References

[1] Brookner, E., ed. *Radar Technology*. Lexington, MA: LexBook, 1996.

[2] Brookner, E., ed. *Practical Phased Array Antenna Systems*. Boston: Artech House, 1991.

[3] Mailloux, R. J. “Phased Array Theory and Technology,” *Proceedings of the IEEE*, Vol., 70, Number 3s, pp. 246–291.

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

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

See Also

`phased.ReplicatedSubarray` | `phased.PartitionedArray` | `phased.ConformalArray` | `phased.CosineAntennaElement` | `phased.CustomAntennaElement` | `phased.IsotropicAntennaElement` | `phased.ULA` | `phased.HeterogeneousULA` | `phased.HeterogeneousURA`

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`.

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`

phased.URA.getElementPosition

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 down each column, then to the next column to the right. For example, in a URA with 4 elements in each row and 3 elements in each column, the element in the third row and second column has an index value of 6.

Examples

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

Array element tapers

Syntax

```
wts = getTaper(h)
```

Description

`wts = getTaper(h)` returns the tapers, `wts`, applied to each element of the phased uniform rectangular array (URA), `h`. Tapers are often referred to as *weights*.

Input Arguments

h - Uniform rectangular array

`phased.URA` System object

Uniform rectangular array specified as `phased.URA` System object.

Output Arguments

wts - Array element tapers

N-by-1 complex-valued vector

Array element tapers returned as an *N*-by-1, complex-valued vector, where *N* is the number of elements in the array.

Examples

URA Array Element Tapering

Construct a 5-by-2 element URA with a Taylor window taper along each column. Then, draw the array showing the element taper shading.

```
taper = taylorwin(5);
ha = phased.URA([5,2],'Taper',[taper,taper]);
w = getTaper(ha)
viewArray(ha,'ShowTaper',true);
```

```
w =

    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
    0.5181
    1.2029
    1.5581
    1.2029
    0.5181
```

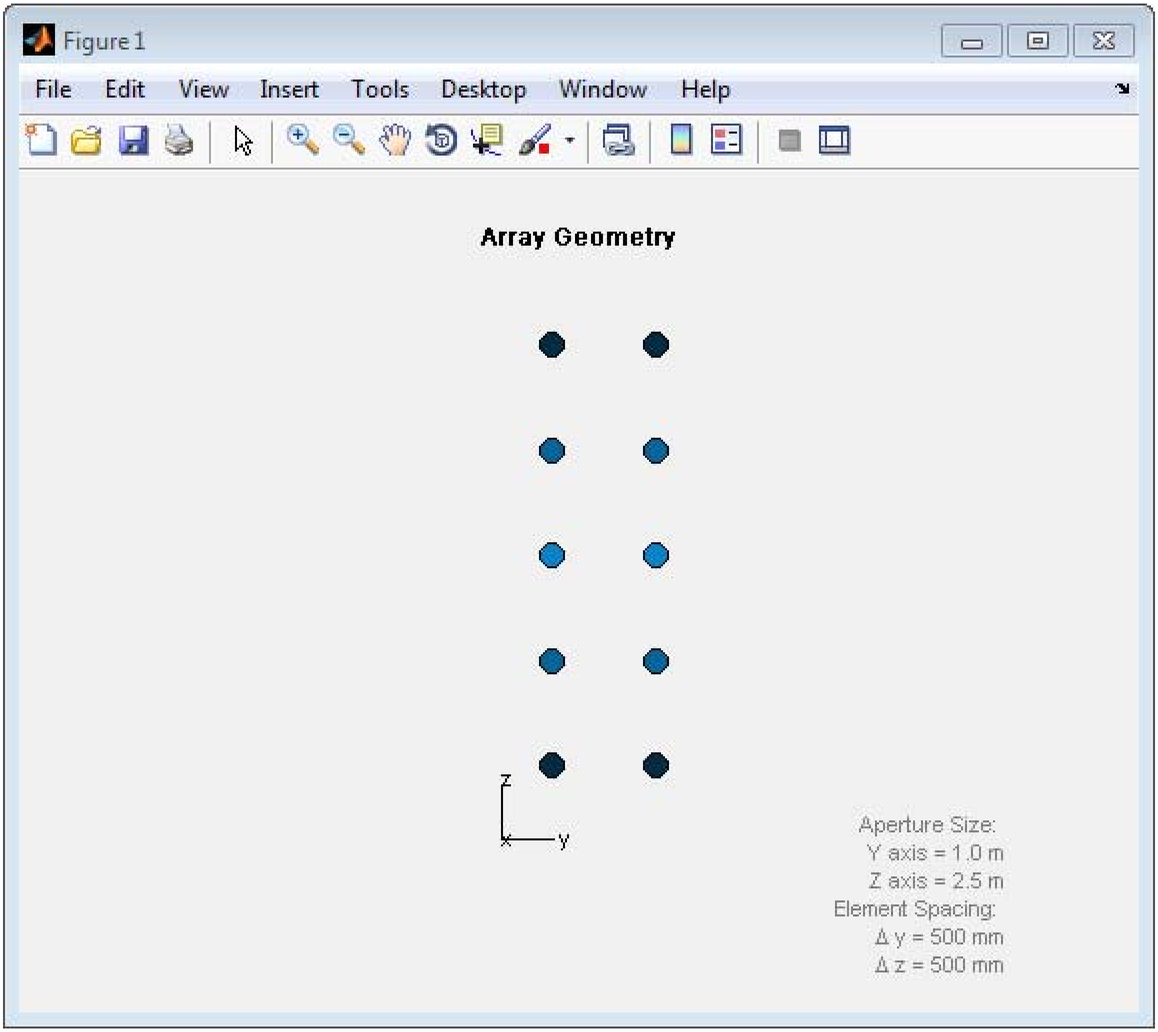


Figure 1
File Edit View Insert Tools Desktop Window Help
Array Geometry
z
y
x
Aperture Size:
Y axis = 1.0 m
Z axis = 2.5 m
Element Spacing:
Δ y = 500 mm
Δ z = 500 mm

phased.URA.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 `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.isPolarizationCapable

Purpose

Polarization capability

Syntax

`flag = isPolarizationCapable(h)`

Description

`flag = isPolarizationCapable(h)` returns a Boolean value, `flag`, indicating whether the array supports polarization. An array supports polarization if all of its constituent sensor elements support polarization.

Input Arguments

h - Uniform rectangular array

Uniform rectangular array specified as `phased.URA` System object.

Output Arguments

flag - Polarization-capability flag

Polarization-capability flag returned as a Boolean value, `true`, if the array supports polarization or, `false`, if it does not.

Examples

Short-Dipole Antenna Array Polarization

Determine whether an array of `phased.ShortDipoleAntennaElement` short-dipole antenna element supports polarization.

```
h = phased.ShortDipoleAntennaElement(...
    'FrequencyRange',[1e9 10e9]);
ha = phased.URA([3,2],'Element',h);
isPolarizationCapable(ha)
```

```
ans =

     1
```

The returned value `true (1)` shows that this array supports polarization.

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 specified as a scalar or 1-by-*K* row vector. Values must lie within the range specified by a property of `H`. That property is named `FrequencyRange` or `FrequencyVector`, depending on the type of element in the array. The element has no response at frequencies outside that range. If you set the `'RespCut'` property of `H` to `'3D'`, `FREQ` must be a scalar. When `FREQ` is a row vector, `plotResponse` draws multiple frequency responses on the same axes.

V

Propagation speed in meters per second.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `true`

'OverlayFreq'

Set this value to `true` to overlay pattern cuts in a 2-D line plot. Set this value to `false` to plot pattern cuts against frequency in a 3-D waterfall plot. If this value is `false`, `FREQ` must be a vector with at least two entries.

This parameter applies only when `Format` is not `'Polar'` and RespCut is not `'3D'`.

Default: `true`

'Polarization'

Specify the polarization options for plotting the array response pattern. The allowable values are |`'None'` | `'Combined'` | `'H'` | `'V'` | where

- `'None'` specifies plotting a nonpolarized response pattern
- `'Combined'` specifies plotting a combined polarization response pattern
- `'H'` specifies plotting the horizontal polarization response pattern
- `'V'` specifies plotting the vertical polarization response pattern

For arrays that do not support polarization, the only allowed value is `'None'`. This parameter is not applicable when you set the `Unit` parameter value to `'dbi'`.

Default: `'None'`

'RespCut'

Cut of the response. Valid values depend on `Format`, as follows:

- If `Format` is `'Line'` or `'Polar'`, the valid values of `RespCut` are `'Az'`, `'El'`, and `'3D'`. The default is `'Az'`.
- If `Format` is `'UV'`, the valid values of `RespCut` are `'U'` and `'3D'`. The default is `'U'`.

If you set `RespCut` to `'3D'`, `FREQ` must be a scalar.

'Unit'

The unit of the plot. Valid values are `'db'`, `'mag'`, `'pow'`, or `'dbi'`. This parameter determines the type of plot that is produced.

Unit value	Plot type
`db`	power pattern in dB scale
`mag`	field pattern
`pow`	power pattern
`dbi`	directivity

Default: `'db'`

'Weights'

Weight values applied to the array, specified as a length-N column vector or N-by-M matrix. The dimension N is the number of elements in the array. The interpretation of M depends upon whether the input argument `FREQ` is a scalar or row vector.

Weights Dimensions	FREQ Dimension	Purpose
N-by-1 column vector	Scalar or 1-by-M row vector	Apply one set of weights for the same single frequency or all M frequencies.
N-by-M matrix	Scalar	Apply all of the M different columns in `Weights` for the same single frequency.
	1-by-M row vector	Apply each of the M different columns in `Weights` for the corresponding frequency in `FREQ`.

'AzimuthAngles'

Azimuth angles for plotting array response, specified as a row vector. The `AzimuthAngles` parameter sets the display range and resolution of azimuth angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'Az'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of azimuth angles should lie between –180° and 180° and must be in nondecreasing order. When you set the `RespCut` parameter to `'3D'`, you can set the `AzimuthAngles` and `ElevationAngles` parameters simultaneously.

Default: `[-180:180]`

'ElevationAngles'

Elevation angles for plotting array response, specified as a row vector. The `ElevationAngles` parameter sets the display range and resolution of elevation angles for visualizing the radiation pattern. This parameter is allowed only when the `RespCut` parameter is set to `'El'` or `'3D'` and the `Format` parameter is set to `'Line'` or `'Polar'`. The values of elevation angles should lie between –90° and 90° and must be in nondecreasing order. When yous set the `RespCut` parameter to `'3D'`, you can set the `ElevationAngles` and `AzimuthAngles` parameters simultaneously.

Default: `[-90:90]`

'UGrid'

U coordinate values for plotting array response, specified as a row vector. The `UGrid` parameter sets the display range and resolution of the *U* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'U'` or `'3D'`. The values of `UGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set the `UGrid` and `VGrid` parameters simultaneously.

Default: `[-1:0.01:1]`

'VGrid'

V coordinate values for plotting array response, specified as a row vector. The `VGrid` parameter sets the display range and resolution of the *V* coordinates for visualizing the radiation pattern in *U*/*V* space. This parameter is allowed only when the `Format` parameter is set to `'UV'` and the `RespCut` parameter is set to `'3D'`. The values of `VGrid` should be between –1 and 1 and should be specified in nondecreasing order. You can set `VGrid` and `UGrid` parameters simultaneously.

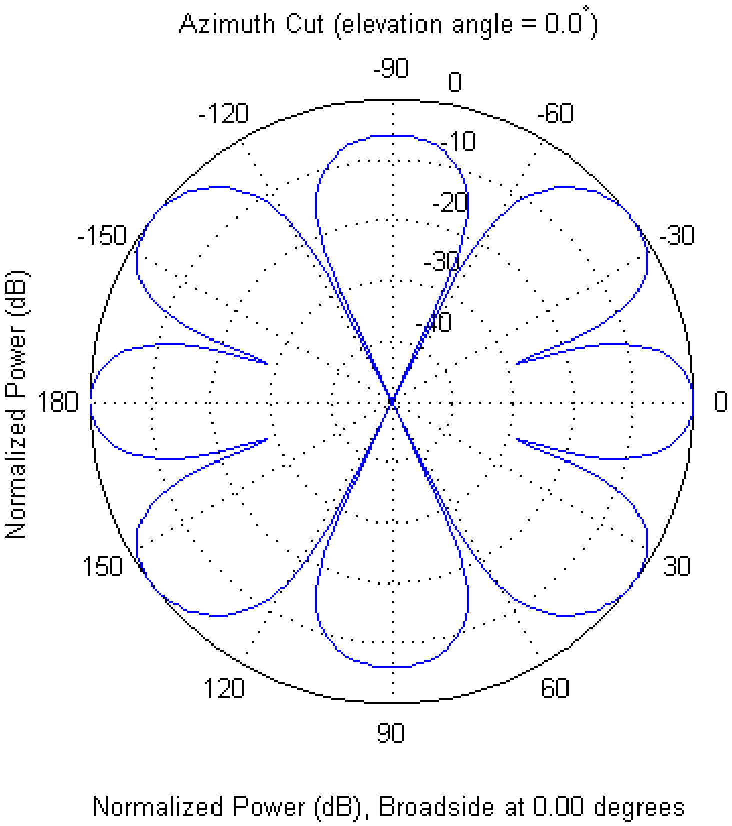
Default: `[-1:0.01:1]`

Examples

Azimuth Response of URA

This example shows how to construct a rectangular lattice 3-by-2 URA and plot that array's azimuth response.

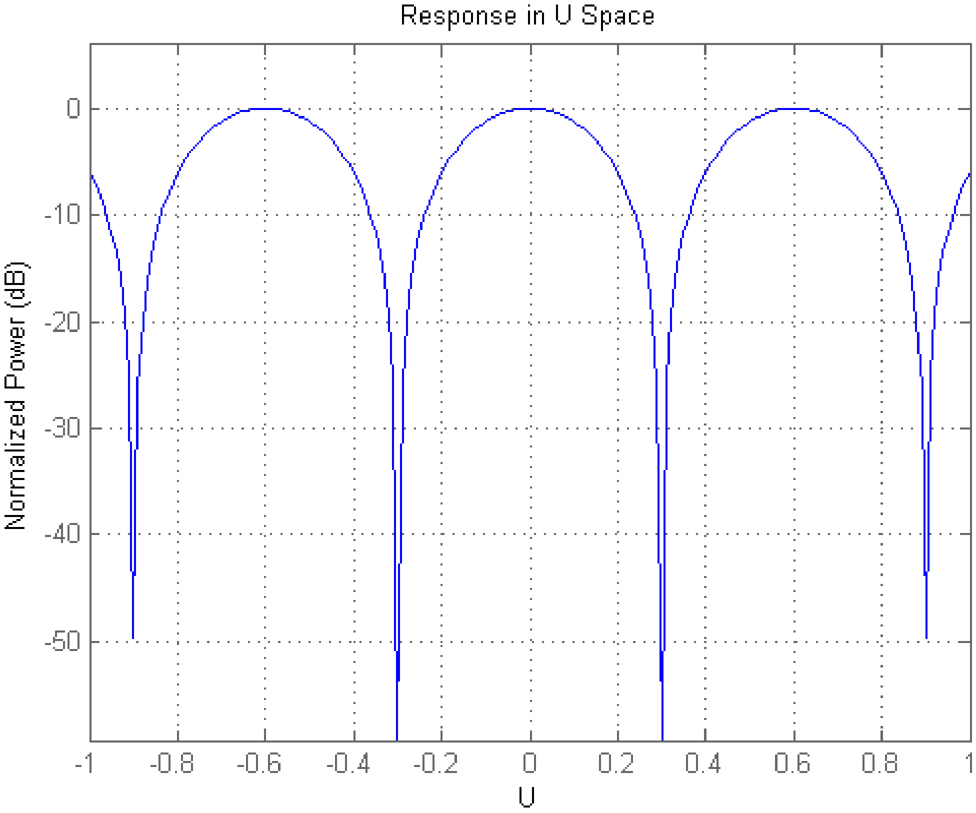
```
ha = phased.URA('Size',[3 2]);
fc = 1e9;
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar');
```



Array Response and Directivity of URA in U/V Space

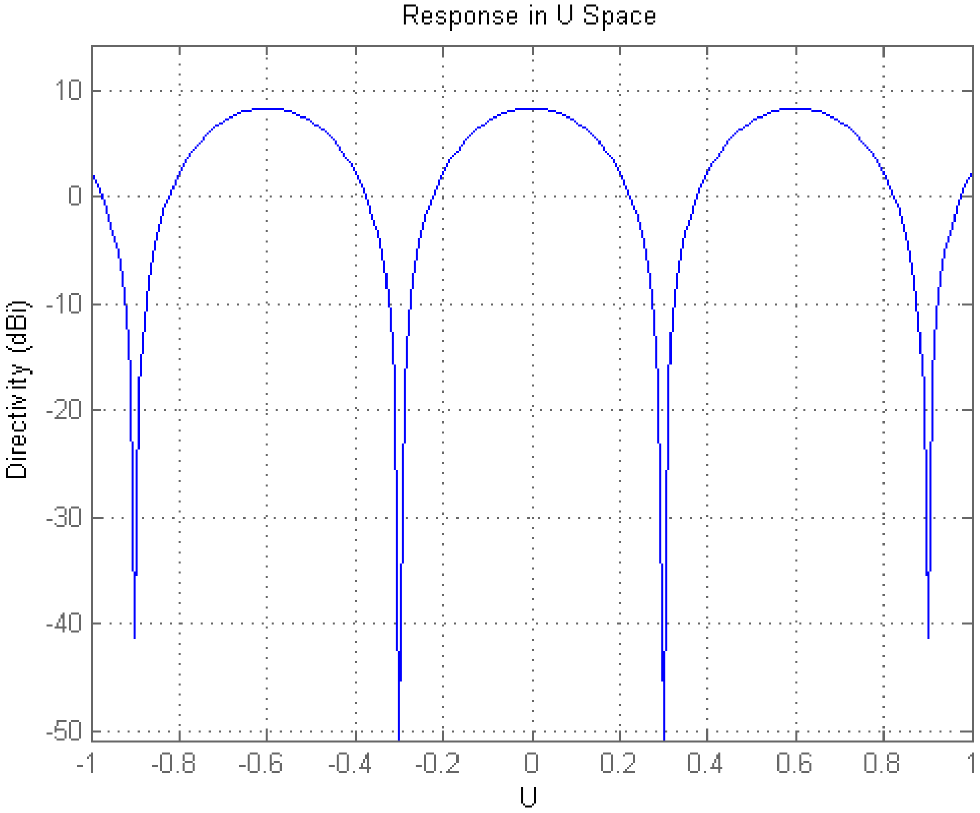
This example shows how to construct a rectangular lattice 3-by-2 URA. Plot the u cut of the array response in $u - v$ space.

```
ha = phased.URA('Size',[3 2]);
c = physconst('lightspeed');
plotResponse(ha,1e9,c,'Format','UV');
```



Plot the directivity.

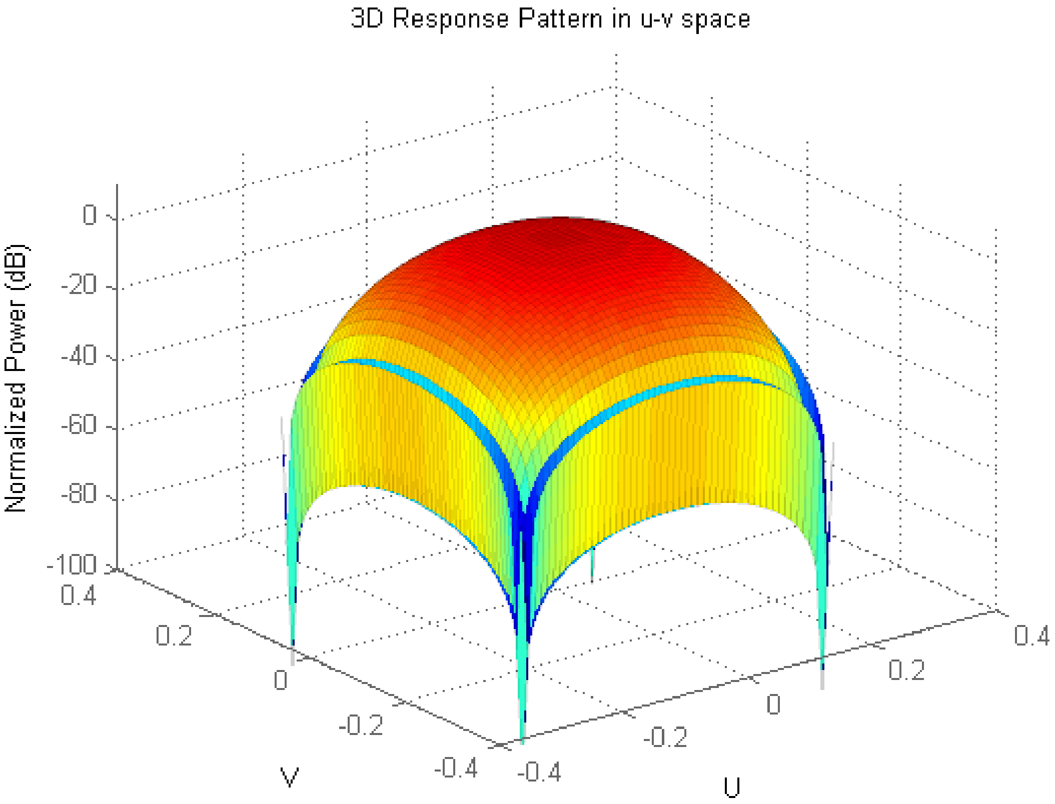
```
plotResponse(ha,1e9,c,'Format','UV','Unit','dbi');
```



Array Response of URA for Subrange of U/V Space

This example shows how to construct a 5-by-5 square URA and plot the 3-D response in u/v space. However, restrict the range in u/v space using the `UGrid` and `VGrid` parameters.

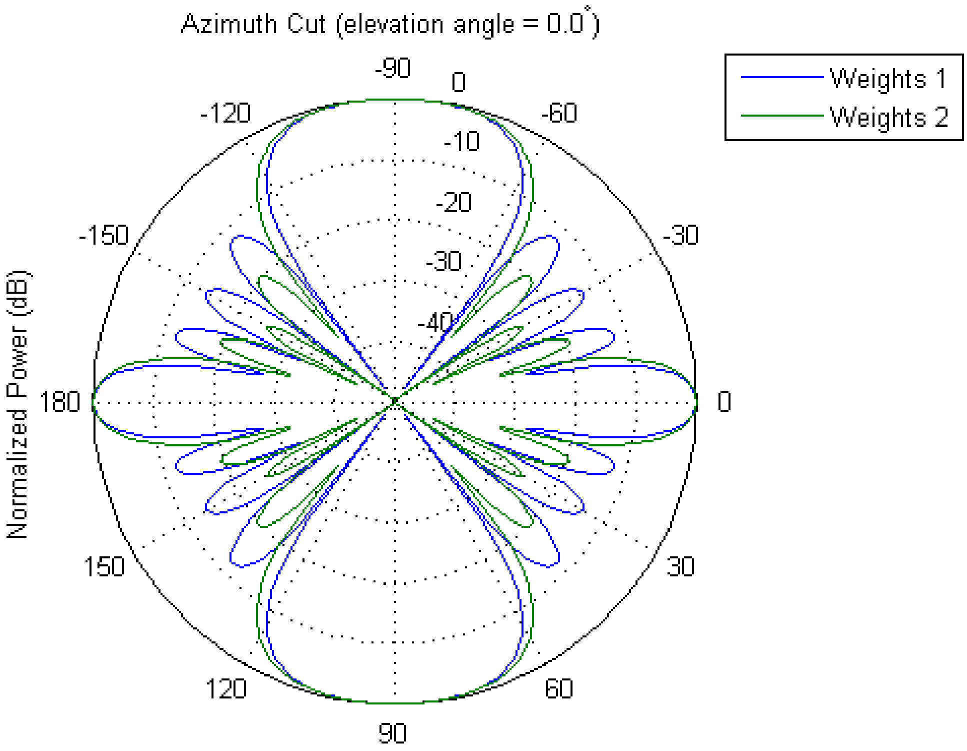
```
ha = phased.URA([5,5]);
fc = 5e8;
c = physconst('LightSpeed');
plotResponse(ha,fc,c,'RespCut','3D','Format','UV',...
'UGrid',[-0.25:.01:.25],'VGrid',[-0.25:.01:.25]);
```



Array Response of URA with Two Sets of Weights

This example shows how to construct a square 5-by-5 URA array having elements spaced 0.3 meters apart. Apply both uniform weights and tapered weights at a single frequency using the `Weights` parameter. Choose the tapered weight values to be smallest at the edges and increasing towards the center. Then, show that the tapered weight set reduces the adjacent sidelobes while broadening the main lobe.

```
ha = phased.URA('Size',[5 5],'ElementSpacing',[0.3,0.3]);
fc = 1e9;
c = physconst('LightSpeed');
wts1 = ones(5,5);
wts1 = wts1(:);
wts1 = wts1/sum(wts1);
wts2 = 0.3*ones(5,5);
wts2(2:4,2:4) = 0.7;
wts2(3,3) = 1;
wts2 = wts2(:);
wts2 = wts2/sum(wts2);
plotResponse(ha,fc,c,'RespCut','Az','Format','Polar','Weights',[wts1,w
```



Normalized Power (dB), Broadside at 0.00 degrees

See Also `uv2azel` | `azel2uv`

Purpose

Plot grating lobe diagram of array

Syntax

```
plotGratingLobeDiagram(H,FREQ)
plotGratingLobeDiagram(H,FREQ,ANGLE)
plotGratingLobeDiagram(H,FREQ,ANGLE,C)
plotGratingLobeDiagram(H,FREQ,ANGLE,C,FO)
hPlot = plotGratingLobeDiagram(___)
```

Description

`plotGratingLobeDiagram(H,FREQ)` plots the grating lobe diagram of an array in the *u-v* coordinate system. The System object `H` specifies the array. The argument `FREQ` specifies the signal frequency and phase-shifter frequency. The array, by default, is steered to 0° azimuth and 0° elevation.

A grating lobe diagram displays the positions of the peaks of the narrowband *array pattern*. The array pattern depends only upon the geometry of the array and not upon the types of elements which make up the array. Visible and nonvisible grating lobes are displayed as open circles. Only grating lobe peaks near the location of the mainlobe are shown. The mainlobe itself is displayed as a filled circle.

`plotGratingLobeDiagram(H,FREQ,ANGLE)`, in addition, specifies the array steering angle, `ANGLE`.

`plotGratingLobeDiagram(H,FREQ,ANGLE,C)`, in addition, specifies the propagation speed by `C`.

`plotGratingLobeDiagram(H,FREQ,ANGLE,C,FO)`, in addition, specifies an array phase-shifter frequency, `FO`, that differs from the signal frequency, `FREQ`. This argument is useful when the signal no longer satisfies the narrowband assumption and, allows you to estimate the size of beam squint.

`hPlot = plotGratingLobeDiagram(___)` returns the handle to the plot for any of the input syntax forms.

Input Arguments

H

Antenna or microphone array, specified as a System object.

FREQ

Signal frequency, specified as a scalar. Frequency units are hertz. Values must lie within a range specified by the frequency property of the array elements contained in `H.Element`. The frequency property is named `FrequencyRange` or `FrequencyVector`, depending on the element type.

ANGLE

Array steering angle, specified as either a 2-by-1 vector or a scalar. If `ANGLE` is a vector, it takes the form `[azimuth;elevation]`. The azimuth angle must lie in the range `[-180 ,180 ]`. The elevation angle must lie in the range `[-90 ,90 ]`. All angle values are specified in degrees. If the argument `ANGLE` is a scalar, it specifies only the azimuth angle where the corresponding elevation angle is 0°.

Default: `[0;0]`

C

Signal propagation speed, specified as a scalar. Units are meters per second.

Default: Speed of light in vacuum

F0

Phase-shifter frequency of the array, specified as a scalar. Frequency units are hertz When this argument is omitted, the phase-shifter frequency is assumed to be the signal frequency, `FREQ`.

Default: `FREQ`

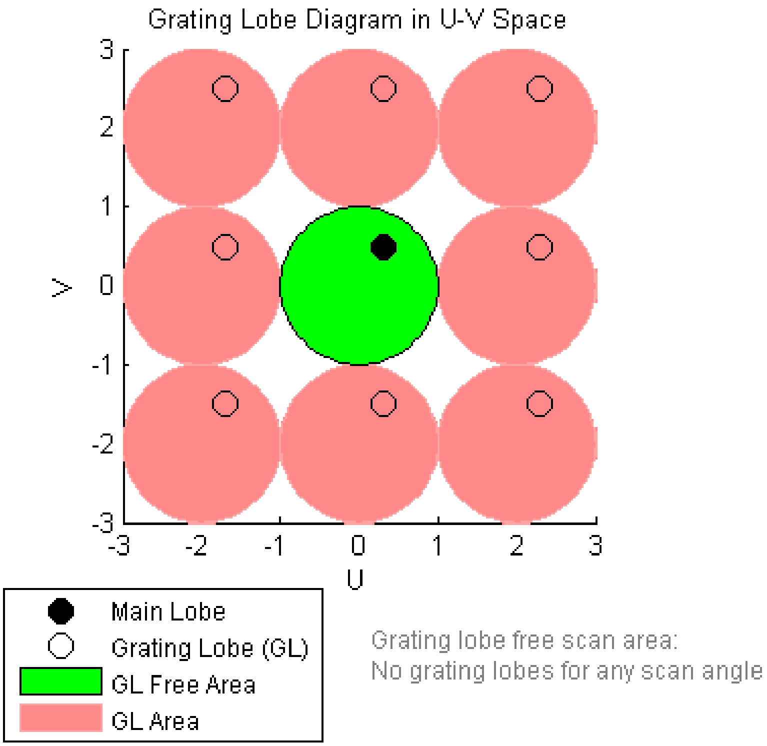
Examples

Create Grating Lobe Diagram for Microphone URA

Plot the grating lobe diagram for an 11-by-9-element uniform rectangular array having element spacing equal to one-half wavelength.

Assume the operating frequency of the array is 10 kHz. All elements are omnidirectional microphone elements. Steer the array in the direction 20 degrees in azimuth and 30 degrees in elevation. The speed of sound in air is 344.21 m/s at 21 deg C.

```
cair = 344.21;
f = 10000;
lambda = cair/f;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
sURA = phased.URA('Element',sMic,'Size',[11,9],...
    'ElementSpacing',0.5*lambda*[1,1]);
plotGratingLobeDiagram(sURA,f,[20;30],cair);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in visible and nonvisible regions are indicated by unfilled black circles. The visible region is the region in u-v coordinates for which $u^2 + v^2 \leq 1$. The visible region is shown as a unit circle centered at the origin. Because the array spacing is less than one-half wavelength, there are no grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it coincides with the visible region.

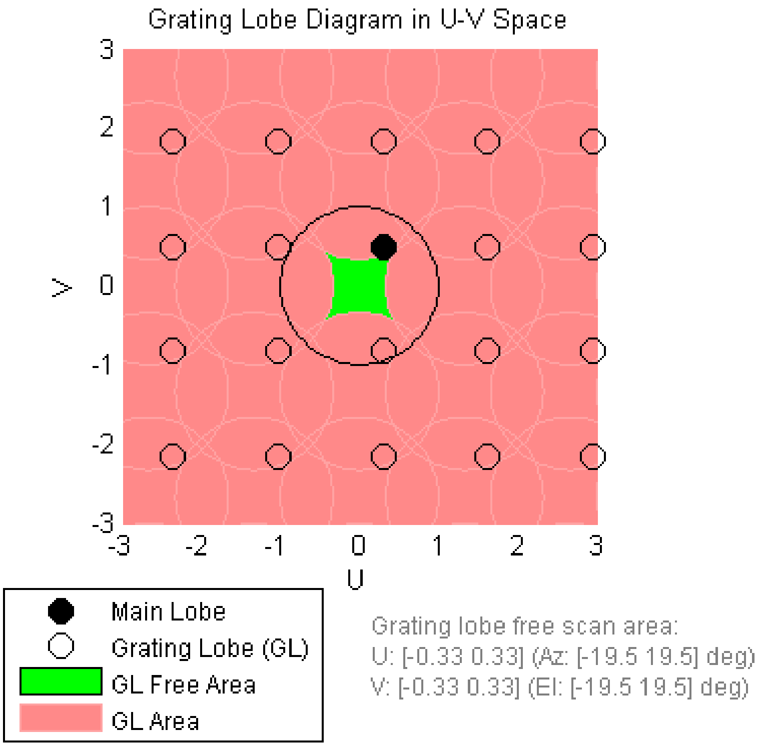
The white areas of the diagram indicate a region where no grating lobes are possible.

Create Grating Lobe Diagram for Undersampled Microphone URA

Plot the grating lobe diagram for an 11-by-9-element uniform rectangular array having element spacing greater than one-half wavelength. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 10 kHz and the spacing between elements is 0.75 of a wavelength. All elements are omnidirectional microphone elements. Steer the array in the direction 20 degrees in azimuth and 30 degrees in elevation. The speed of sound in air is 344.21 m/s at 21 deg C.

```
cair = 344.21;
f = 10000;
lambda = cair/f;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
sURA = phased.URA('Element',sMic,'Size',[11,9],...
    'ElementSpacing',0.75*lambda*[1,1]);
plotGratingLobeDiagram(sURA,f,[20;30],cair);
```



The main lobe of the array is indicated by a filled black circle. The grating lobes in visible and nonvisible regions are indicated by unfilled black circles. The visible region is the region in u-v coordinates for which $u^2 + v^2 \leq 1$. The visible region is shown as a unit circle centered at the origin. Because the array spacing is greater than one-half wavelength, there are grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

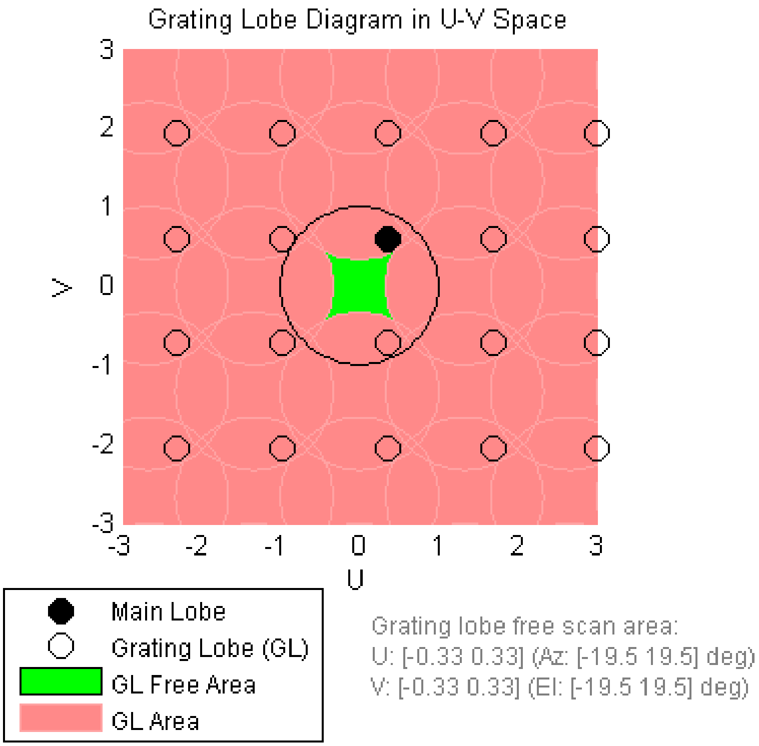
The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region. Because the mainlobe is outside the green area, there is a grating lobe within the visible region.

Create Grating Lobe Diagram for Microphone URA with Frequency Shift

Plot the grating lobe diagram for an 11-by-9-element uniform rectangular array having element spacing greater than one-half wavelength. Apply a 20% phase-shifter frequency offset. Grating lobes are plotted in u-v coordinates.

Assume the operating frequency of the array is 10 kHz and the spacing between elements is 0.75 of a wavelength. All elements are omnidirectional microphone elements. Steer the array in the direction 20 degrees in azimuth and 30 degrees in elevation. The shifted frequency is 12000 Hz. The speed of sound in air is 344.21 m/s at 21 deg C.

```
cair = 344.21;
f = 10000;
f0 = 12000;
lambda = cair/f;
sMic = phased.OmnidirectionalMicrophoneElement(...
    'FrequencyRange',[20 20000]);
sURA = phased.URA('Element',sMic,'Size',[11,9],...
    'ElementSpacing',0.75*lambda*[1,1]);
plotGratingLobeDiagram(sURA,f,[20;30],cair,f0);
```



The mainlobe of the array is indicated by a filled black circle. The mainlobe has moved from its position in the previous example due to the frequency shift. The grating lobes in visible and nonvisible regions are indicated by unfilled black circles. The visible region is the region in u-v coordinates for which $u^2 + v^2 \leq 1$. The visible region is shown as a unit circle centered at the origin. Because the array spacing is greater than one-half wavelength, there are grating lobes in the visible region of space. There are an infinite number of grating lobes in the nonvisible regions, but only those in the range [-3,3] are shown.

The grating-lobe free region, shown in green, is the range of directions of the main lobe for which there are no grating lobes in the visible region. In this case, it lies inside the visible region. Because the mainlobe is outside the green area, there is a grating lobe within the visible region.

Concepts

Grating Lobes

Spatial undersampling of a wavefield by an array gives rise to visible grating lobes. If you think of the wavenumber, k, as analogous to angular frequency, then you must sample the signal at spatial intervals smaller than π/k_{max} (or $\lambda_{min}/2$) in order to remove aliasing. The appearance of visible grating lobes is also known as spatial aliasing. The variable k_{max} is the largest wavenumber value present in the signal.

The directions of maximum spatial response of a URA are determined by the peaks of the array's *array pattern* (alternatively called the *beam pattern* or *array factor*). Peaks other than the mainlobe peak are called grating lobes. For a URA, the array pattern depends only on the wavenumber component of the wavefield in the array plane (the y and z directions for the `phased.URA` System object). The wavenumber components are related to the look-direction of an arriving wavefield by $k_y = -2\pi \sin az \cos el/\lambda$ and $k_z = -2\pi \sin el/\lambda$. The angle az is azimuth angle of the arriving wavefield. The angle el is elevation angle of the arriving wavefield. The look-direction points away from the array to the wavefield source.

The array pattern possesses an infinite number of periodically-spaced peaks that are equal in strength to the mainlobe peak. If you steer the array to the az_0, el_0 azimuth and elevation direction, the array pattern for the URA has its mainlobe peak at the wavenumber value, $k_{y0} = -2\pi \sin az_0 \cos el_0/\lambda$, $k_{z0} = -2\pi \sin el_0/\lambda$. The array pattern has strong peaks at $k_{ym} = k_{y0} + 2\pi\, m/d_y$, $k_{zn} = k_{z0} + 2\pi\, n/d_z$. for integral values of m and n. The quantities d_y and d_z are the inter-element spacings in the y- and z-directions, respectively. Expressed in terms of direction cosines, the grating lobes occur at $u_m = u_0 - m\lambda/d_y$ and $v_n = v_0 - n\lambda/d_z$. The mainlobe direction cosines are given by $u_0 = \sin az_0 \cos el_0$ and $v_0 = \sin el_0$ when expressed in terms of the look-direction.

Grating lobes can be visible or nonvisible, depending upon the value of $u_m^2 + v_n^2$. When $u_m^2 + v_n^2 \leq 1$, the look direction represent a visible direction. When the value is greater than one, the grating lobe is nonvisible. For each visible grating lobe, you can compute a look direction $(az_{m,n},el_{m,n})$ from $u_m = \sin az_m \cos el_m$ and $v_n = \sin el_n$. The spacing of grating lobes depends upon λ/d. When λ/d is small enough, multiple grating lobe peaks can correspond to physical look-directions.

References

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

See Also

`uv2azel` | `azel2uv`

phased.URA.release

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

Voltage responses of the phased array. The output depends on whether the array supports polarization or not.

- If the array is not capable of supporting polarization, the voltage response, `RESP`, has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array. The dimension *M* is the number of angles specified in `ANG`. *L* is the number of frequencies specified in `FREQ`. For any element, the columns of `RESP` contain the responses of the array elements for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.
- If the array is capable of supporting polarization, the voltage response, `RESP`, is a MATLAB `struct` containing two fields, `RESP.H` and `RESP.V`. The field, `RESP.H`, represents the array's horizontal polarization response, while `RESP.V` represents the array's vertical polarization response. Each field has the dimensions *N*-by-*M*-by-*L*. *N* is the number of elements in the array, and *M* is the number of angles specified in `ANG`. *L* is the number of frequencies specified in `FREQ`. Each column of `RESP` contains the responses of the array elements for the corresponding direction specified in `ANG`. Each of the *L* pages of `RESP` contains the responses of the array elements for the corresponding frequency specified in `FREQ`.

Examples

Response of a 2-by-2 URA of Short-Dipole Antennas

Construct a 2-by-2 rectangular lattice URA of short-dipole antenna elements. Then, find the response of each element at boresight. Assume the operating frequency is 1 GHz.

```
h = phased.ShortDipoleAntennaElement;
ha = phased.URA('Element',h,'Size',[2 2]);
fc = 1e9; ang = [0;0];
resp = step(ha,fc,ang);
disp(resp.V);
```

```
   -1.2247
   -1.2247
   -1.2247
   -1.2247
```

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. `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`

'ShowTaper'

Set this value to `true` to specify whether to change the element color brightness in proportion to the element taper magnitude. When this value is set to `false`, all elements are drawn with the same color.

Default: `false`

'Title'

String specifying the title of the plot.

Default: `'Array Geometry'`

Output Arguments

hPlot

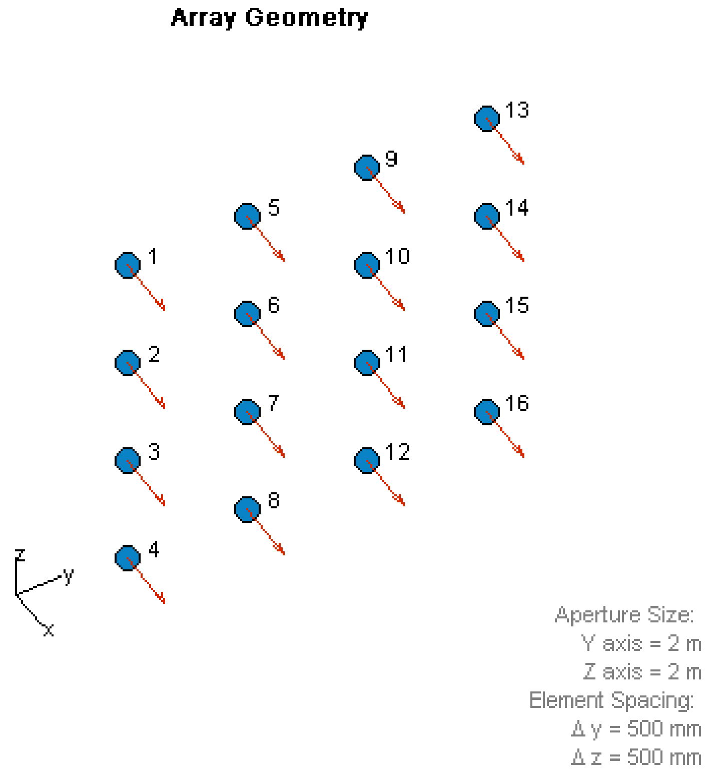
Handle of array elements in figure window.

Examples

Geometry, Normal Directions, and Indices of URA Elements

This example shows how to display the element positions, normal directions, and indices for all elements of a 4-by-4 square URA.

```
ha = phased.URA(4);
viewArray(ha,'ShowNormals',true,'ShowIndex','All');
```


Array Geometry
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
z
y
x
Aperture Size:
Y axis = 2 m
Z axis = 2 m
Element Spacing:
Δ y = 500 mm
Δ z = 500 mm

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 1-1258.
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

Sensor element or sensor array

Sensor element or sensor array specified as a System object in the Phased Array System Toolbox. A sensor array can contain subarrays.

Default: `phased.ULA` with default property values

PropagationSpeed

Signal propagation speed

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

Default: Speed of light

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`

EnablePolarization

EnablePolarization

Set this property to `true` to simulate the collection of polarized waves. Set this property to `false` to ignore polarization. This property applies when the sensor specified in the `Sensor` property is capable of simulating polarization.

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`

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.

phased.WidebandCollector.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.WidebandCollector.step

Purpose

Collect signals

Syntax

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

Description

`Y = step(H,X,ANG)` collects signals `X` arriving from directions `ANG`. The collection process depends on the `Wavefront` property of `H`, as follows:

- If `Wavefront` has the value `'Plane'`, each collecting element collects all the far field signals in `X`. Each column of `Y` contains the output of the corresponding element in response to all the signals in `X`.
- If `Wavefront` has the value `'Unspecified'`, each collecting element collects only one impinging signal from `X`. Each column of `Y` contains the output of the corresponding element in response to the corresponding column of `X`. The `'Unspecified'` option is available when the `Sensor` property of `H` does not contain subarrays.

`Y = step(H,X,ANG,LAXES)` uses `LAXES` as the local coordinate system axes directions. This syntax is available when you set the `EnablePolarization` property to `true`.

`Y = step(H,X,ANG,WEIGHTS)` uses `WEIGHTS` as the weight vector. This syntax is available when you set the `WeightsInputPort` property to `true`.

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

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

Note The object performs an initialization the first time the `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.

- If the `EnablePolarization` property value is set to `false`, `X` is a matrix. The number of columns of the matrix equals the number of separate signals.
- If the `EnablePolarization` property value is set to `true`, `X` is a row vector of MATLAB `struct` type. The dimension of the `struct` array equals the number of separate signals. Each

`struct` member contains three column-vector fields, `X`, `Y`, and `Z`, representing the *x*, *y*, and *z* components of the polarized wave vector signals in the global coordinate system.

ANG

Incident directions of signals, specified as a two-row matrix. Each column specifies the incident direction of the corresponding column of `X`. Each column of `ANG` has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

LAXES

Local coordinate system. `LAXES` is a 3-by-3 matrix whose columns specify the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively. Each axis is specified in terms of `[x;y;z]` with respect to the global coordinate system. This argument is only used when the `EnablePolarization` property is set to `true`.

WEIGHTS

Vector of weights. `WEIGHTS` is a column vector of length M, where M is the number of collecting elements.

Default: `ones(M,1)`

STEERANGLE

Subarray steering angle, specified as a length-2 column vector. The vector has the form [azimuth; elevation], in degrees. The azimuth angle must be between –180 and 180 degrees, inclusive. The elevation angle must be between –90 and 90 degrees, inclusive.

Output Arguments

Y

Collected signals. Each column of `Y` contains the output of the corresponding element. The output is the response to all the signals in `X`, or one signal in `X`, depending on the `Wavefront` property of `H`.

Examples

Collect 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.

2

Functions-Alphabetical List

aictest

Purpose

Dimension of signal subspace

Syntax

```
nsig = aictest(X)
nsig = aictest(X,'fb')
```

Description

`nsig = aictest(X)` estimates the number of signals, `nsig`, present in a *snapshot* of data, `X`, that impinges upon the sensors in an array. The estimator uses the *Akaike Information Criterion test* (AIC). The input argument, `X`, is a complex-valued matrix containing a time sequence of data samples for each sensor. Each row corresponds to a single time sample for all sensors.

`nsig = aictest(X,'fb')` estimates the number of signals. Before estimating, it performs *forward-backward averaging* on the sample covariance matrix constructed from the data snapshot, `X`. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

X - Data snapshot

Complex-valued K-by-N matrix

Data snapshot, specified as a complex-valued, K-by-N matrix. A snapshot is a sequence of time-samples taken simultaneous at each sensor. In this matrix, K represents the number of time samples of the data, while N represents the number of sensor elements.

Example: [–0.1211 + 1.2549i, 0.1415 + 1.6114i, 0.8932 + 0.9765i;]

Data Types
`double`
Complex Number Support: Yes

Output Arguments

nsig - Dimension of signal subspace

Non-negative integer

Dimension of signal subspace, returned as a non-negative integer. The dimension of the signal subspace is the number of signals in the data.

Examples

Estimate the Signal Subspace Dimensions for Two Arriving Signals

Construct a data snapshot for two plane waves arriving at a half-wavelength-spaced uniform line array with 10 elements. The plane waves arrive from 0° and –25° azimuth, both with elevation angles of 0°. Assume the signals arrive in the presence of additive noise that is both temporally and spatially Gaussian white noise. For each signal, the SNR is 5 dB. Take 300 samples to build a 300-by-10 data snapshot. Then, solve for the number of signals using `aictest`.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25];
x = sensorsig(elementPos,300,angles,db2pow(-5));
Nsig = aictest(x)

Nsig =

     2
```

The result shows that the number of signals is two, as expected.

Estimate the Signal Subspace Dimension with Forward-Backward Smoothing

Construct a data snapshot for two plane waves arriving at a half-wavelength-spaced uniform line array with 10 elements. Correlated plane waves arrive from 0° and 10° azimuth, both with elevation angles of 0°. Assume the signals arrive in the presence of additive noise that is both temporally and spatially Gaussian white noise. For each signal, the SNR is 10 dB. Take 300 samples to build a 300-by-10 data snapshot. Then, solve for the number of signals using `aictest`.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
```

```
angles = [0 10];
ncov = db2pow(-10);
scov = [1 .5]'*[1 .5];
x = sensorsig(elementPos,300,angles,ncov,scov);
Nsig = aictest(x)

Nsig =

     1
```

This result shows that `aictest` function cannot determine the number of signals correctly when the signals are correlated.

Now, try the option of forward-backward smoothing.

```
Nsig = aictest(x,'fb')

Nsig =

     2
```

The addition of forward-backward smoothing yields the correct number of signals.

Definitions

Estimating the Number of Sources

AIC and MDL tests

Direction finding algorithms such as MUSIC and ESPRIT require knowledge of the number of sources of signals impinging on the array or equivalently, the dimension, d, of the signal subspace. The Akaike Information Criterion (AIC) and the Minimum Description Length (MDL) formulas are two frequently-used estimators for obtaining that dimension. Both estimators assume that, besides the signals, the data contains spatially and temporally white Gaussian random noise. Finding the number of sources is equivalent to finding the multiplicity of the smallest eigenvalues of the sampled spatial covariance matrix. The sample spatial covariance matrix constructed from a data snapshot is used in place of the actual covariance matrix.

A requirement for both estimators is that the dimension of the signal subspace be less than the number of sensors, *N*, and that the number of time samples in the snapshot, *K*, be much greater than *N*.

A variant of each estimator exists when forward-backward averaging is employed to construct the spatial covariance matrix. Forward-backward averaging is useful for the case when some of the sources are highly correlated with each other. In that case, the spatial covariance matrix may be ill conditioned. Forward-backward averaging can only be used for certain types of symmetric arrays, called *centro-symmetric* arrays. Then the forward-backward covariance matrix can be constructed from the sample spatial covariance matrix, *S*, using $S_{FB} = S + JS^{*}J$ where *J* is the exchange matrix. The exchange matrix maps array elements into their symmetric counterparts. For a line array, it would be the identity matrix flipped from left to right.

All the estimators are based on a cost function

$$L_d(d) = K(N-d)\ln\left\{\frac{\frac{1}{N-d}\sum_{i=d+1}^{N}\lambda_i}{\left\{\prod_{i=d+1}^{N}\lambda_i\right\}^{\frac{1}{N-d}}}\right\}$$

plus an added penalty term. The value λ_i represent the smallest *(N–d)* eigenvalues of the spatial covariance matrix. For each specific estimator, the solution for *d* is given by

- AIC

$$\hat{d}_{AIC} = \underset{d}{\operatorname{argmin}}\{L_d(d) + d(2N-d)\}$$

- AIC for forward-backward averaged covariance matrices

$$\hat{d}_{AIC:FB} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + \frac{1}{2}d(2N-d+1)\right\}$$

- MDL

$$\hat{d}_{MDL} = \underset{d}{\operatorname{argmin}} \left\{ L_d(d) + \frac{1}{2}(d(2N-d)+1)\ln K \right\}$$

- MDL for forward-backward averaged covariance matrices

$$\hat{d}_{MDL\,FB} = \underset{d}{\operatorname{argmin}} \left\{ L_d(d) + \frac{1}{4}d(2N-d+1)\ln K \right\}$$

References

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

See Also

`espritdoa` | `mdltest` | `rootmusicdoa` | `spsmooth`

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)');
```

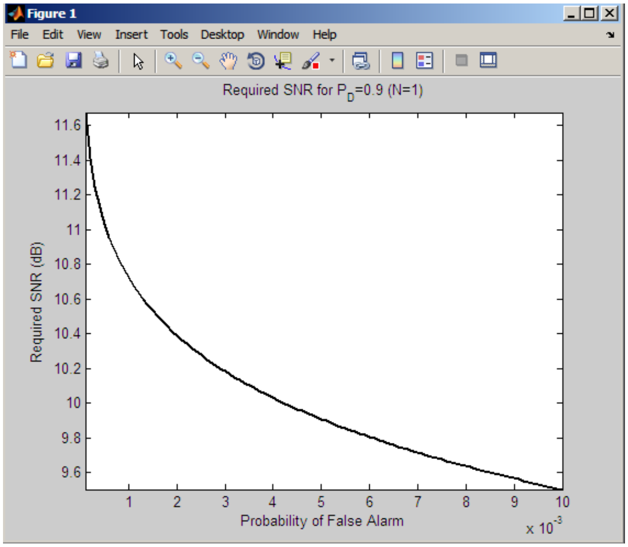


Figure 1
File Edit View Insert Tools Desktop Window Help
Required SNR for P_D=0.9 (N=1)
11.6
11.4
11.2
11
10.8
10.6
10.4
10.2
10
9.8
9.6
Required SNR (dB)
1
2
3
4
5
6
7
8
9
10
Probability of False Alarm
x 10^{-3}

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)');
```

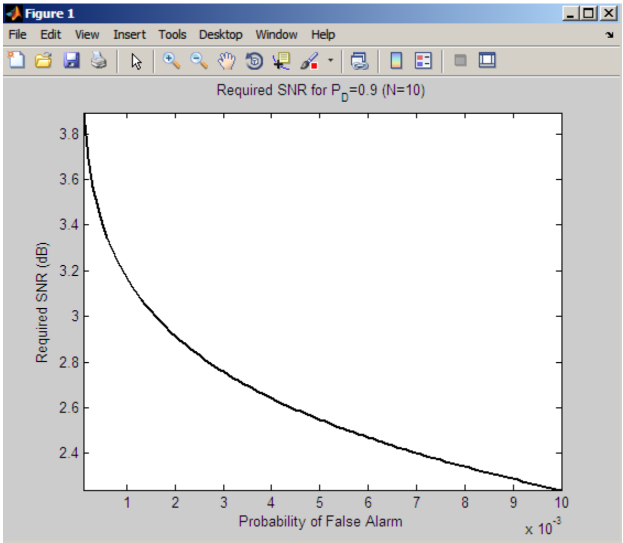


Figure 1
File Edit View Insert Tools Desktop Window Help
Required SNR for P_D=0.9 (N=10)
Required SNR (dB)
3.8
3.6
3.4
3.2
3
2.8
2.6
2.4
1 2 3 4 5 6 7 8 9 10
Probability of False Alarm
x 10^{-3}

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')
[afmag,delay] =
ambgfun(x,Fs,PRF,'Cut','Doppler','CutValue',V)
[afmag,doppler] =
ambgfun(x,Fs,PRF,'Cut','Delay','CutValue',V)
ambgfun(x,Fs,PRF)
ambgfun(x,Fs,PRF,'Cut','2D')
ambgfun(x,Fs,PRF,'Cut','Delay')
ambgfun(x,Fs,PRF,'Cut','Doppler')
ambgfun(x,Fs,PRF,'Cut','Delay','CutValue',V)
ambgfun(x,Fs,PRF,'Cut','Doppler','CutValue',V)
```

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.

`[afmag,delay] = ambgfun(x,Fs,PRF,'Cut','Doppler','CutValue',V)` returns a one-dimensional cut through the 2-D normalized ambiguity function magnitude at a Doppler value of `V` Hertz. `V` should lie in the range`[-Fs/2,Fs/2]`.

`[afmag,doppler] = ambgfun(x,Fs,PRF,'Cut','Delay','CutValue',V)` returns a one-dimensional cut through the 2-D normalized ambiguity function magnitude at a delay value of `V` seconds. `V` should lie in the range `[-(length(x)-1)/Fs,(length(x)-1)/Fs]`.

`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.

`ambgfun(x,Fs,PRF,'Cut','Delay','CutValue',V)` or `ambgfun(x,Fs,PRF,'Cut','Doppler','CutValue',V)` with no output argument produces a line plot of the ambiguity function cut at non-zero cut values.

Input Arguments

x

Pulse waveform. `x` is a row or column vector.

Fs

Sampling frequency in hertz.

PRF

Pulse repetition frequency in hertz.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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`.

Example: `'Cut','Doppler','CutValue',10` specifies that a vector of ambiguity function values be produced at a Doppler shift of 10 Hz.

'Cut' - Direction of one-dimensional cut through ambiguity function

`'Delay'|'Doppler'|'2D'`

Used to generate a one–dimensional cut or cross-section through the ambiguity diagram. The direction of the cut is determined by setting the value of `'Cut'` to `'Delay'` or `'Doppler'`. The value `'2D'` generates a surface plot of the two-dimensional ambiguity function.

The choice of `'Delay'` generates a cut at zero time delay. In this case, the second output argument of `ambgfun` contains the ambiguity function values at Doppler shifted values. A cut at non-zero time delay can be generated using the name-value pair`'CutValue'` described below.

The choice of `'Doppler'` generates a cut at zero Doppler shift. In this case, the second output argument of `ambgfun` contains the ambiguity function values at time-delayed values. A cut at non-zero Doppler can be generated using the name-value pair `'CutValue'` described below.

'CutValue' - Optional time delay or Doppler shift at which ambiguity function cut is taken

real scalar

When setting the name-value pair `'Cut'` to `'Delay'` or `'Doppler'`, you can use the name-value pair `'CutValue'` to specify a cross-section that does not coincide with either zero time delay or zero Doppler shift. However, `'CutValue'` should not be used when `'Cut'` is set to `'2D'.`

When `'Cut'` is set to `'Delay'` ,`'CutValue'` is interpreted as the time delay, in seconds, at which the cut is to be taken. The range of possible time delays is determined by the length of the signal and is restricted to `[-(length(x)-1)/Fs,(length(x)-1)/Fs]`.

When `'Cut'` is set to `'Doppler'`, `'CutValue'` is interpreted as the Doppler shift, in Hertz, at which the cut is to be taken. The Doppler shift is restricted to the range `[-Fs/2,Fs/2]`.

Example: `'CutValue',10.0`

Data Types
`double`

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.

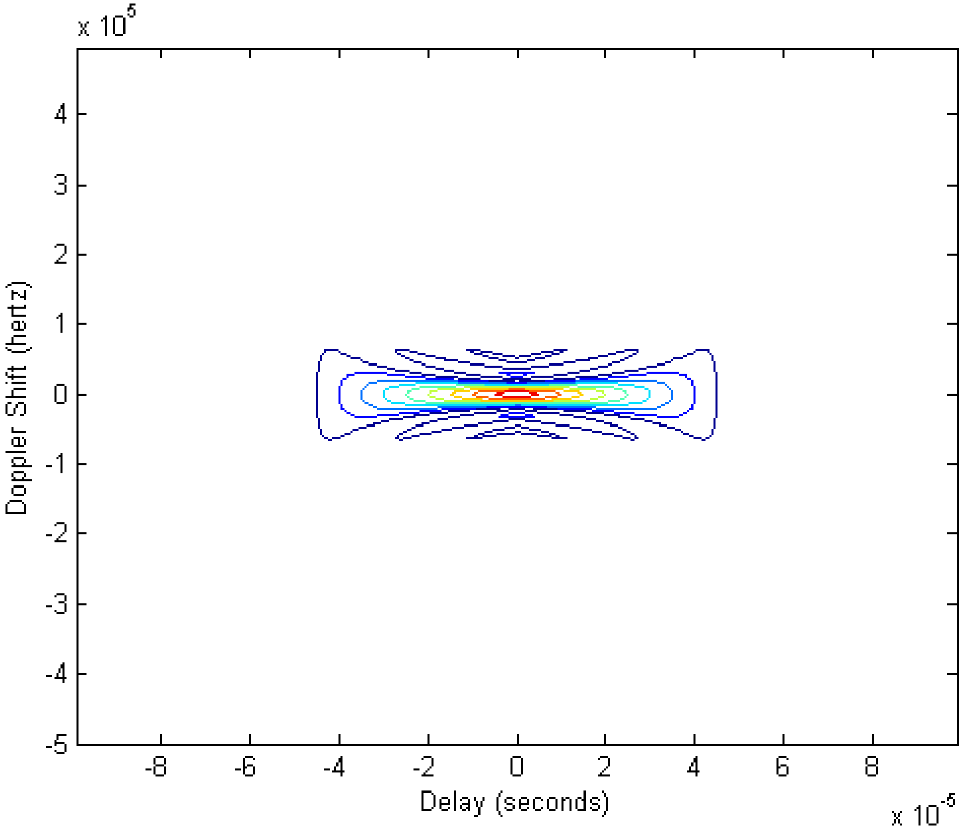
The magnitude of the ambiguity function achieves maximum value at (0,0). At this point, there is perfect correspondence between the received waveform and the matched filter. In the normalized ambiguity function, the maximum value equals one.

Examples

Plot Ambiguity Function of Rectangular Pulse

Plot the ambiguity function magnitude of a rectangular pulse.

```
hrect = phased.RectangularWaveform;
x = step(hrect);
PRF = 2e4;
[afmag,delay,doppler] = ambgfun(x,hrect.SampleRate,PRF);
contour(delay,doppler,afmag);
xlabel('Delay (seconds)'); ylabel('Doppler Shift (hertz)');
```

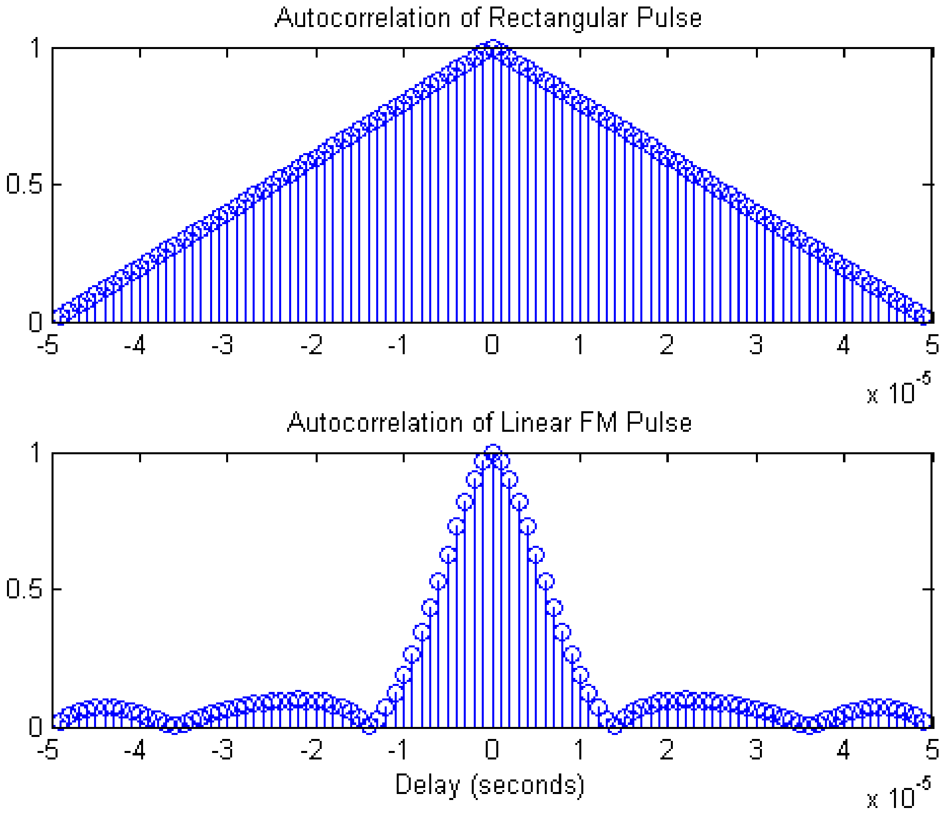


Plot Autocorrelation Sequences of Rectangular and Linear FM Pulses

This example shows how to plot zero-Doppler cuts of the autocorrelation sequences of rectangular and linear FM pulses of equal duration. Note the pulse compression exhibited in the autocorrelation sequence of the linear FM pulse.

```
hrect = phased.RectangularWaveform('PRF',2e4);
hfm = phased.LinearFMWaveform('PRF',2e4);
```

```
xrect = 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');
```

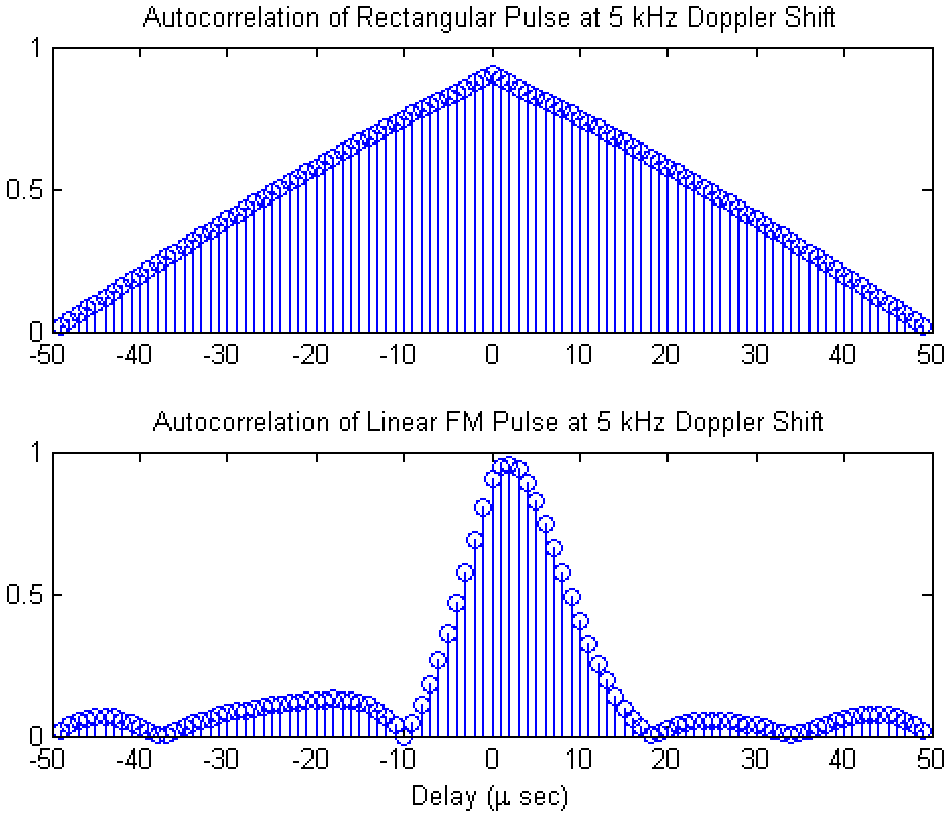


Plot Nonzero-Doppler Cuts of Autocorrelation Sequences

Plot nonzero-Doppler cuts of the autocorrelation sequences of rectangular and linear FM pulses of equal duration. Both cuts are taken at a 5 kHz Doppler shift. Besides the reduction of the peak value, there is a strong shift in the position of the linear FM peak, evidence of range-doppler coupling.

```
hrect = phased.RectangularWaveform('PRF',2e4);
hfm = phased.LinearFMWaveform('PRF',2e4);
```

```
xrect = step(hrect);
xfm = step(hfm);
fd = 5000;
[ambrect,delayrect] = ambgfun(xrect,hrect.SampleRate,...,
    hrect.PRF,'Cut','Doppler','CutValue',fd);
[ambfm,delayfm] = ambgfun(xfm,hfm.SampleRate,...,
    hfm.PRF,'Cut','Doppler','CutValue',fd);
figure;
subplot(211);
stem(delayrect*10^6,ambrect);
title('Autocorrelation of Rectangular Pulse at 5 kHz Doppler Shift');
subplot(212);
stem(delayfm*10^6,ambfm)
xlabel('Delay (\mu sec)');
title('Autocorrelation of Linear FM Pulse at 5 kHz Doppler Shift');
```



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`

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`

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 $-180 \le az \le 180$ and $-90 \le el \le 90$.

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`.

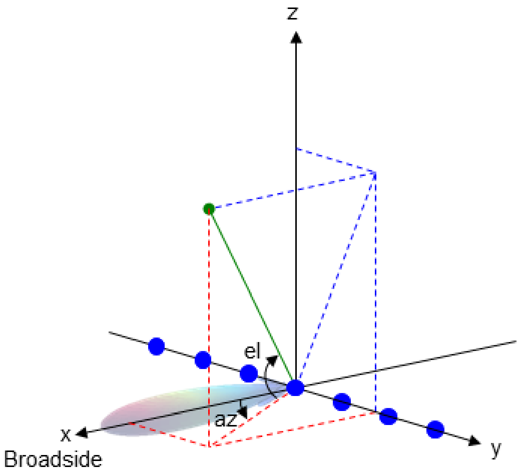
Definitions

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.

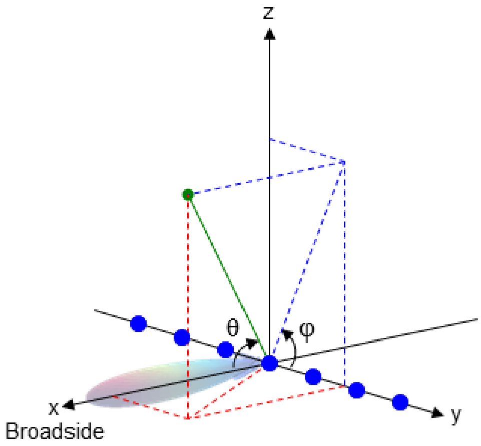
This figure illustrates the azimuth angle and elevation angle for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi\sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi\tan\theta$$

$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el}) / \sin(\mathrm{az})$$

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"

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.

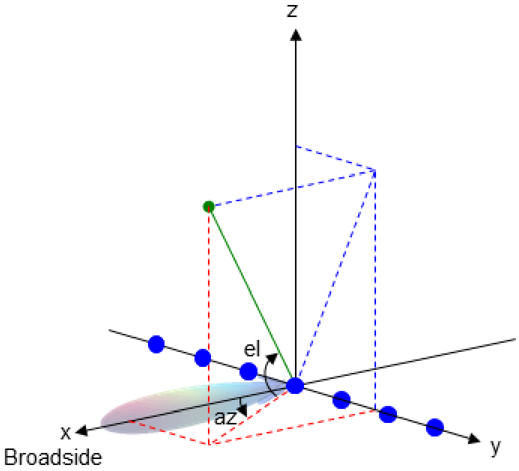
Definitions

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.

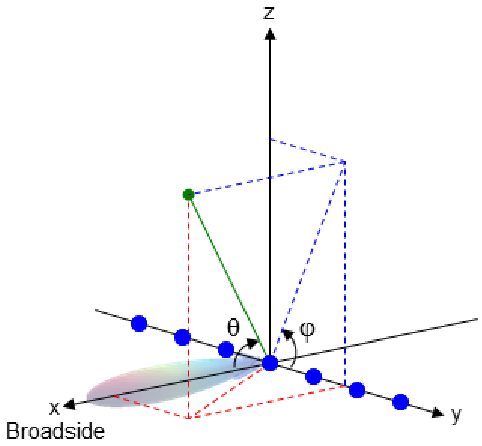
This figure illustrates the azimuth angle and elevation angle for a vector that appears as a green solid line. The coordinate system is relative to the center of a uniform linear array, whose elements appear as blue circles.



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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi \sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi \tan\theta$$

$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el}) / \sin(\mathrm{az})$$

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 Converted Radiation Pattern

Plot the result of converting a radiation pattern to ϕ/θ space with the ϕ and θ angles spaced 1 degree apart.

The radiation pattern is the cosine of the elevation.

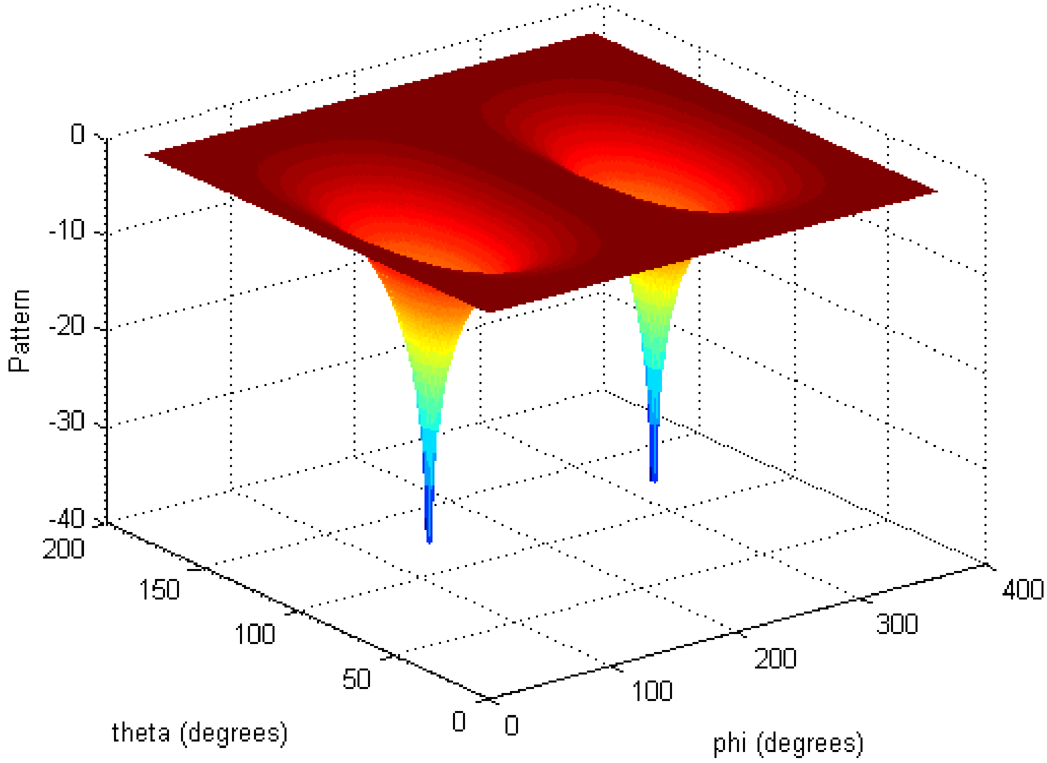
```
az = -180:180;
el = -90:90;
pat_azel = repmat(cosd(el)',1,numel(az));
```

Convert the pattern to ϕ/θ space. Use the returned ϕ and θ angles for plotting.

```
[pat_phitheta,phi,theta] = azel2phithetapat(pat_azel,az,el);
```

Plot the result.

```
H = surf(phi,theta,mag2db(pat_phitheta));
set(H,'LineStyle','none')
xlabel('phi (degrees)');
ylabel('theta (degrees)');
zlabel('Pattern');
```



Convert Radiation Pattern For Specific Phi/Theta Values

Convert a radiation pattern to ϕ/θ space with ϕ and θ angles spaced 5 degrees apart.

The radiation pattern is the cosine of the elevation.

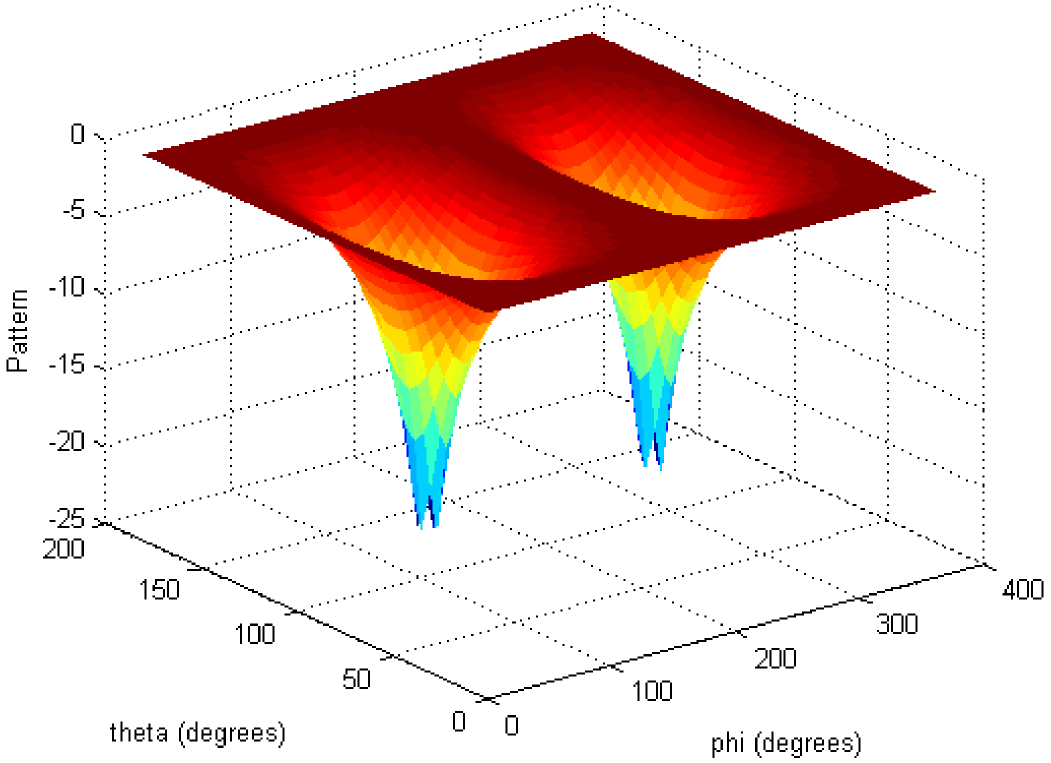
```
az = -180:180;
el = -90:90;
pat_azel = repmat(cosd(el)',1,numel(az));
```

Define the set of ϕ and θ angles at which to sample the pattern. Then, convert the pattern.

```
phi = 0:5:360;
theta = 0:5:180;
pat_phitheta = azel2phithetapat(pat_azel,az,el,phi,theta);
```

Plot the result.

```
H = surf(phi,theta,mag2db(pat_phitheta));
set(H,'LineStyle','none')
xlabel('phi (degrees)');
ylabel('theta (degrees)');
zlabel('Pattern');
```



See Also `phased.CustomAntennaElement` | `phitheta2azel` | `azel2phitheta` | `phitheta2azelpat`

Concepts

- "Spherical Coordinates"

azel2uv

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`.

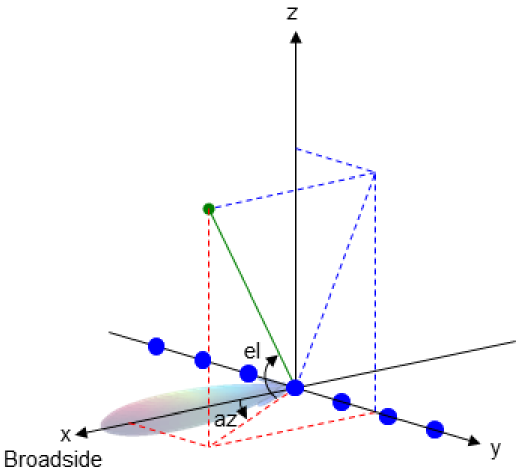
Definitions

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.

In terms of azimuth and elevation, the *u* and *v* coordinates are

$u = \cos(\mathrm{el}) \sin(\mathrm{az})$

$v = \sin(\mathrm{el})$

The values of u and v satisfy the inequalities

$-1 \leq u \leq 1$

$-1 \leq v \leq 1$

$u^2 + v^2 \leq 1$

Conversely, the phi and theta angles can be written in terms of u and v

$\tan(\varphi) = v/u$

$\sin(\theta) = \mathrm{sqrt}(u^2 + v^2)$

The azimuth and elevation angles can also be written in terms of u and v

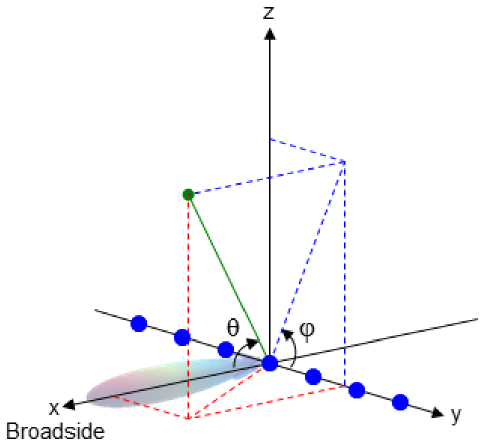
$\sin(\mathrm{el}) = v$

$\tan(\mathrm{az}) = u/\mathrm{sqrt}(1 - u^2 - v^2)$

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.



The coordinate transformations between φ/θ and *az*/*el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi \sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi \tan\theta$$

$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el}) / \sin(\mathrm{az})$$

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”

azel2uvpat

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.

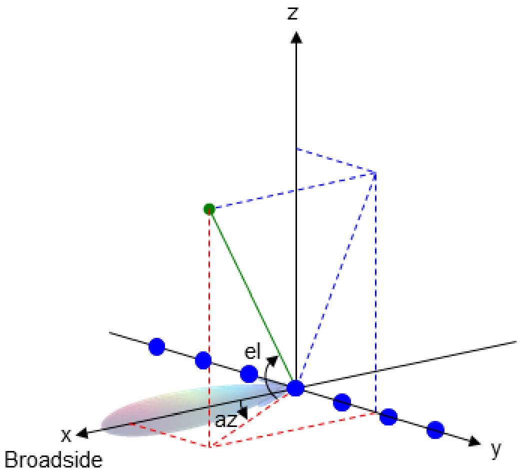
Definitions

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* and *v* coordinates are the direction cosines of a vector with respect to the *y*-axis and *z*-axis, respectively.

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles by:

$$u = \sin(\theta)\cos(\varphi)$$

$$v = \sin(\theta)\sin(\varphi)$$

In these expressions, φ and θ are the phi and theta angles, respectively.

In terms of azimuth and elevation, the *u* and *v* coordinates are

$$u = \cos(\mathrm{el})\sin(\mathrm{az})$$

$$v = \sin(\mathrm{el})$$

The values of u and v satisfy the inequalities

$-1 \leq u \leq 1$

$-1 \leq v \leq 1$

$u^2 + v^2 \leq 1$

Conversely, the phi and theta angles can be written in terms of u and v

$\tan(\varphi) = v/u$

$\sin(\theta) = \mathrm{sqrt}(u^2 + v^2)$

The azimuth and elevation angles can also be written in terms of u and v

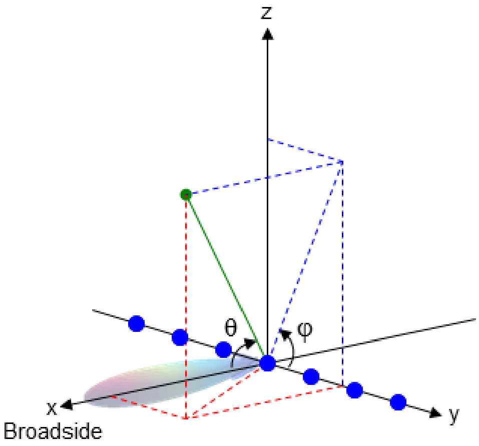
$\sin(\mathrm{el}) = v$

$\tan(\mathrm{az}) = u/\mathrm{sqrt}(1 - u^2 - v^2)$

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.



The coordinate transformations between φ/θ and *az*/*el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi \sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi \tan\theta$$

$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el}) / \sin(\mathrm{az})$$

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 Converted Radiation Pattern

Plot the result of converting a radiation pattern to u/v space with the u and v coordinates spaced by 0.01.

The radiation pattern is the cosine of the elevation angle.

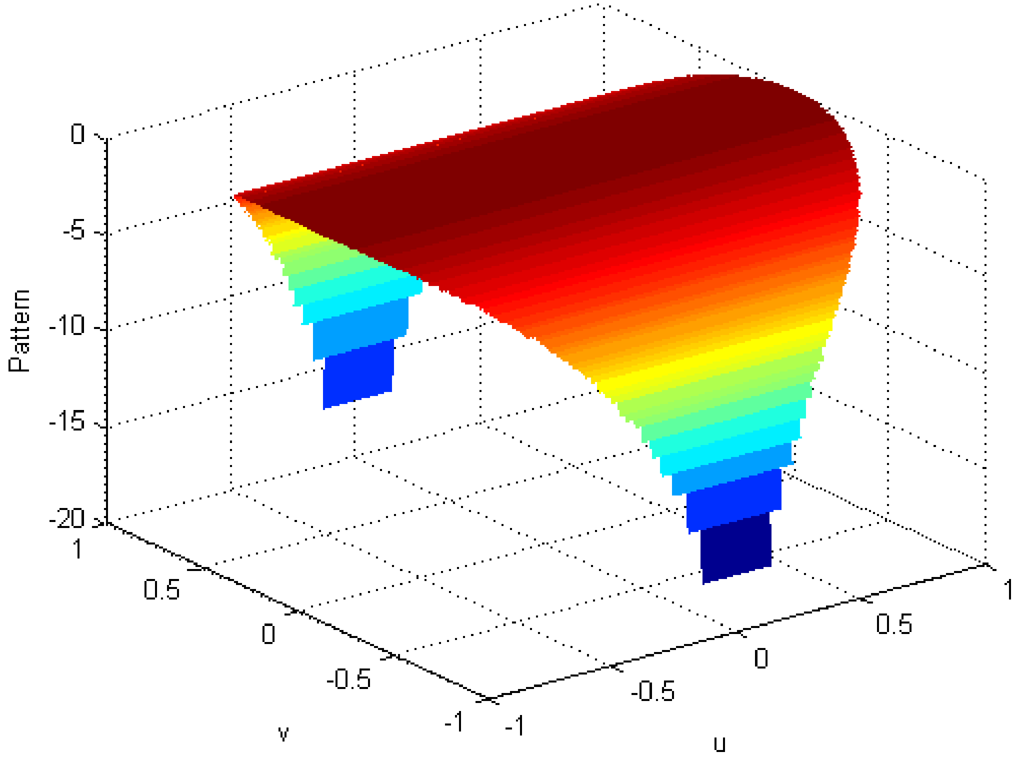
```
az = -90:90;
el = -90:90;
pat_azel = repmat(cosd(el)',1,numel(az));
```

Convert the pattern to u/v space. Use the u and v coordinates for plotting.

```
[pat_uv,u,v] = azel2uvpat(pat_azel,az,el);
```

Plot the result.

```
H = surf(u,v,mag2db(pat_uv));
set(H,'LineStyle','none')
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



Convert Radiation Pattern For Specific U/V Values

Convert a radiation pattern to u/v form, with the u and v coordinates spaced by 0.05.

The radiation pattern is cosine of the elevation angle.

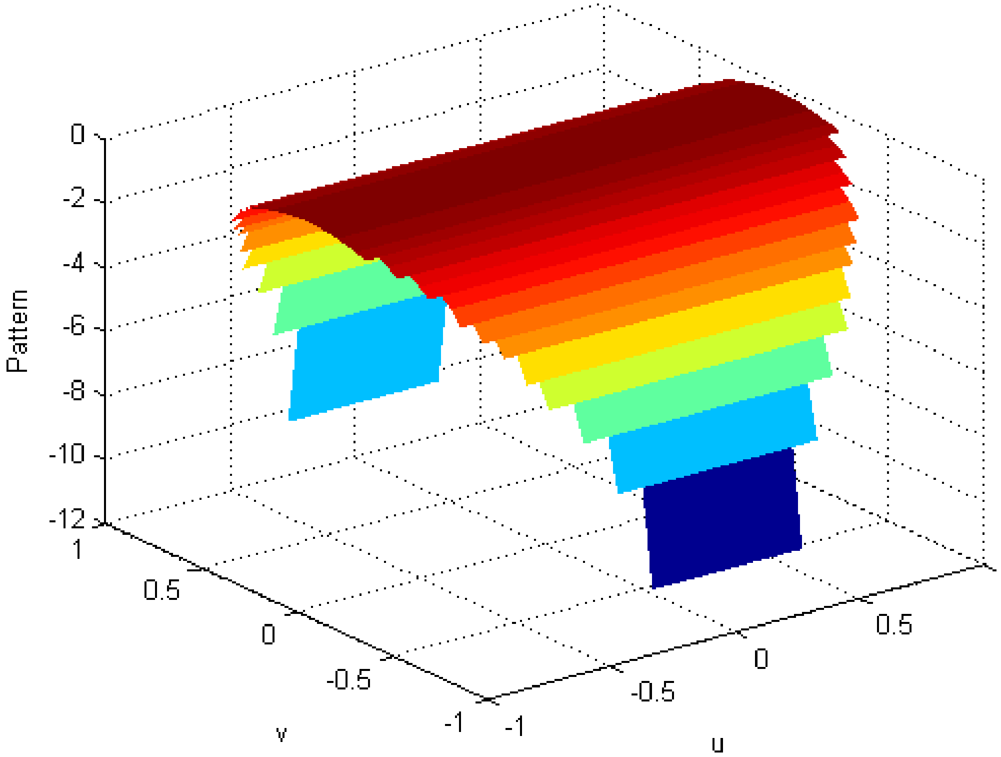
```
az = -90:90;
el = -90:90;
pat_azel = repmat(cosd(el)',1,numel(az));
```

Define the set of `u` and `v` coordinates at which to sample the pattern. Then, convert the pattern.

```
u = -1:0.05:1;
v = -1:0.05:1;
pat_uv = azel2uvpat(pat_azel,az,el,u,v);
```

Plot the result.

```
H = surf(u,v,mag2db(pat_uv));
set(H,'LineStyle','none')
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



See Also `phased.CustomAntennaElement` | `azel2uv` | `uv2azel` | `uv2azelpat`

Concepts

- "Spherical Coordinates"

azelaxes

Purpose

Spherical basis vectors in 3-by-3 matrix form

Syntax

`A = azelaxes(az,el)`

Description

`A = azelaxes(az,el)` returns a 3-by-3 matrix containing the components of the basis $(\mathbf{e}_R, \mathbf{e}_{az}, \mathbf{e}_{el})$ at each point on the unit sphere specified by azimuth, `az`, and elevation, `el`. The columns of `A` contain the components of basis vectors in the order of radial, azimuthal and elevation directions.

Input Arguments

az - Azimuth angle

scalar in range [–180,180]

Azimuth angle specified as a scalar in the closed range [–180,180]. Angle units are in degrees. To define the azimuth angle of a point on a sphere, construct a vector from the origin to the point. The azimuth angle is the angle in the *xy*-plane from the positive *x*-axis to the vector's orthogonal projection into the *xy*-plane. As examples, zero azimuth angle and zero elevation angle specify a point on the *x*-axis while an azimuth angle of 90° and an elevation angle of zero specify a point on the *y*-axis.

Example: `45`

Data Types
`double`

el - Elevation angle

scalar in range [–90,90]

Elevation angle specified as a scalar in the closed range [–90,90]. Angle units are in degrees. To define the elevation of a point on the sphere, construct a vector from the origin to the point. The elevation angle is the angle from its orthogonal projection into the *xy*-plane to the vector itself. As examples, zero elevation angle defines the equator of the sphere and ±90° elevation define the north and south poles, respectively.

Example: `30`

Data Types

`double`

Output Arguments

A - Spherical basis vectors

3-by-3 matrix

Spherical basis vectors returned as a 3-by-3 matrix. The columns contain the unit vectors in the radial, azimuthal, and elevation directions, respectively. Symbolically we can write the matrix as

$$(\mathbf{e}_R, \mathbf{e}_{az}, \mathbf{e}_{el})$$

where each component represents a column vector.

Examples

Spherical Basis Vectors at (45°,45°)

At the point located at 45° azimuth, 45° elevation, compute the 3-by-3 matrix containing the components of the spherical basis:

```
A = azelaxes(45,45)
```

```
A =

    0.5000   -0.7071   -0.5000
    0.5000    0.7071   -0.5000
    0.7071         0    0.7071
```

The first column of `A` is the radial basis vector `[0.5000; 0.5000; 0.7071]`. The second and third columns are the azimuth and elevation basis vectors, respectively.

Algorithms

MATLAB computes the matrix `A` from the equations

```
A = [cosd(el)*cosd(az), -sind(az), -sind(el)*cosd(az); ...
  cosd(el)*sind(az),  cosd(az), -sind(el)*sind(az); ...
  sind(el),           0,         cosd(el)];
```

Definitions

Spherical basis

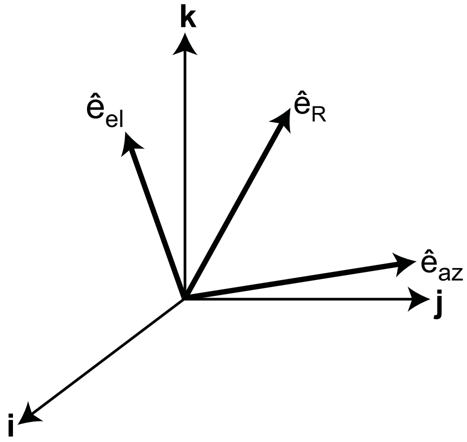
The spherical basis vectors $(\mathbf{e}_R, \mathbf{e}_{az}, \mathbf{e}_{el})$ at the point *(az,el)* can be expressed in terms of the Cartesian unit vectors by

$$\begin{aligned}
\hat{\mathbf{e}}_{\mathbf{R}} &= \cos(el)\cos(az)\hat{\mathbf{i}} + \cos(el)\sin(az)\hat{\mathbf{j}} + \sin(el)\hat{\mathbf{k}} \\
\hat{\mathbf{e}}_{\mathbf{az}} &= -\sin(az)\hat{\mathbf{i}} + \cos(az)\hat{\mathbf{j}} \\
\hat{\mathbf{e}}_{\mathbf{el}} &= -\sin(el)\cos(az)\hat{\mathbf{i}} - \sin(el)\sin(az)\hat{\mathbf{j}} + \cos(el)\hat{\mathbf{k}}
\end{aligned}.$$

This set of basis vectors can be derived from the local Cartesian basis by two consecutive rotations: first by rotating the Cartesian vectors around the *y*-axis by the negative elevation angle, *-el*, followed by a rotation around the *z*-axis by the azimuth angle, *az.* Symbolically, we can write

$$\begin{aligned}
\hat{\mathbf{e}}_{\mathbf{R}} &= R_z(az)R_y(-el)\begin{bmatrix}1\\0\\0\end{bmatrix} \\
\hat{\mathbf{e}}_{\mathbf{az}} &= R_z(az)R_y(-el)\begin{bmatrix}0\\1\\0\end{bmatrix} \\
\hat{\mathbf{e}}_{\mathbf{el}} &= R_z(az)R_y(-el)\begin{bmatrix}0\\0\\1\end{bmatrix}
\end{aligned}$$

The following figure shows the relationship between the spherical basis and the local Cartesian unit vectors.


k
$\hat{e}_{el}$
$\hat{e}_R$
$\hat{e}_{az}$
j
i

See Also `cart2sphvec` | `sph2cartvec`

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)');
```

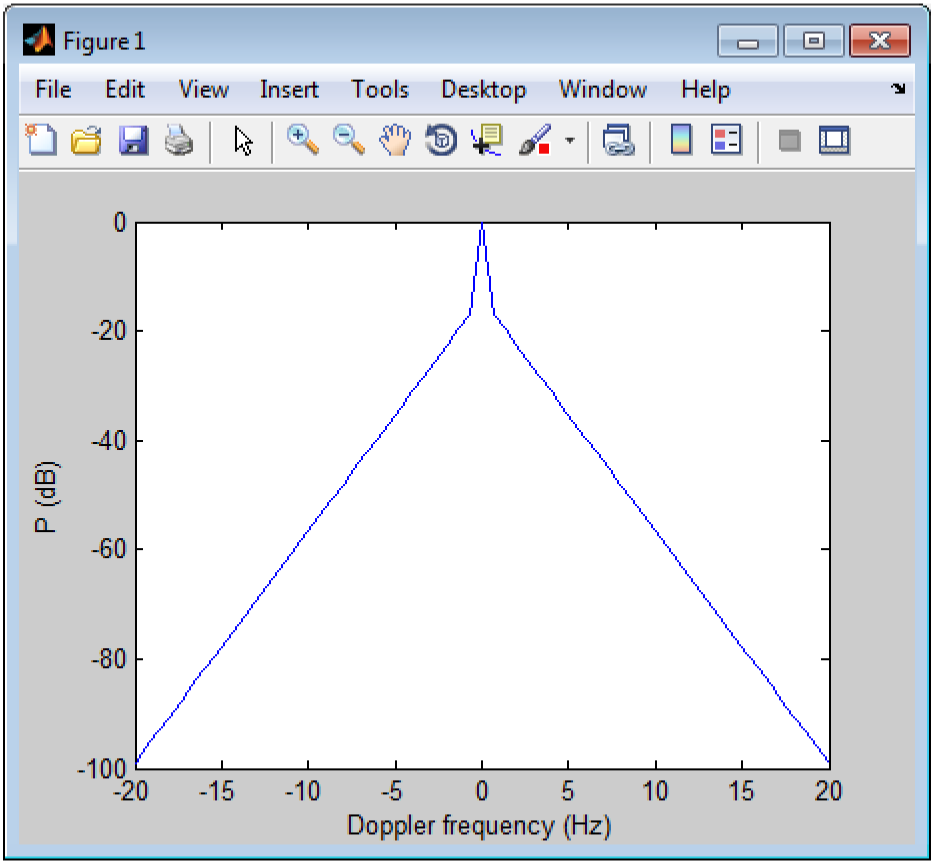


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
0
-20
-40
-60
-80
-100
P (dB)
-20
-15
-10
-5
0
5
10
15
20
Doppler frequency (Hz)

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.

blakechart

Purpose

Range-angle-height (Blake) chart

Syntax

```
blakechart(vcp,vcpangles)
blakechart(vcp,vcpangles,rmax,hmax)
blakechart(___,'Name','Value')
```

Description

`blakechart(vcp,vcpangles)` creates a range-angle-height plot (also called a Blake chart) for a narrowband radar antenna. This chart shows the maximum radar range as a function of target elevation. In addition, the Blake chart displays lines of constant range and lines of constant height. The input consist of the vertical coverage pattern, `vcp`, and vertical coverage pattern angles, `vcpangles`, produced by `radarvcd`.

`blakechart(vcp,vcpangles,rmax,hmax)`, in addition, specifies the maximum range and height of the Blake chart. You can specify range and height units separately in the Name-Value pairs, `RangeUnit` and `HeightUnit`. This syntax can use any of the input arguments in the previous syntax.

`blakechart(___,'Name','Value')` allows you to specify additional input parameters in the form of Name-Value pairs. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). This syntax can use any of the input arguments in the previous syntaxes.

Input Arguments

vcp - Vertical coverage pattern

real-valued vector

Vertical coverage pattern specified as a K-by-1 column vector. The vertical coverage pattern is the actual maximum range of the radar. Each entry of the vertical coverage pattern corresponds to one of the angles specified in `vcpangles`. Values are expressed in kilometers unless you change the unit of measure using the `'RangeUnit'` Name-Value pair.

Example: [282.3831; 291.0502; 299.4252]

Data Types
`double`

vcpangles - Vertical coverage pattern angles

real-valued vector

Vertical coverage pattern angles specified as a *K*-by-1 column vector. The set of angles range from –90° to 90°.

Example: [2.1480; 2.2340; 2.3199]

Data Types
`double`

rmax - Maximum range of plot

real-valued scalar

Maximum range of plot specified as a real-valued scalar. Range units are specified by the `RangeUnit` Name-Value pair.

Example: 200

Data Types
`double`

hmax - Maximum height of plot

real-valued scalar

Maximum height of plot specified as a real-valued scalar. Height units are specified by the `HeightUnit` Name-Value pair.

Example: 100000

Data Types
`double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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`.

Example: ‘RangeUnit’,’m’

’RangeUnit’ - Radar range units

`'km'` (default) | `'nmi'` | `'mi'` | `'ft'` | `'m'`

Range units denoting nautical miles, miles, kilometers, feet or meters. This Name-Value pair specifies the units for the vertical coverage pattern input argument, `vcp`, and the maximum range input argument, `rmax`.

Example: `'mi'`

Data Types

`char`

’HeightUnit’ - Height units

`'km'` (default) | `'nmi'` | `'mi'` | `'ft'` | `'m'`

Height units specified as one of `'nmi'` | `'mi'` | `'km'` | `'ft'` |`'m'` denoting nautical miles, miles, kilometers, feet or meters. This Name-Value pair specifies the units for the maximum height, `hmax`.

Example: `'m'`

Data Types

`char`

’ScalePower’ - Scale power

0.25 (default) | real-valued scalar

Scale power, specified as a scalar between 0 and 1. This parameter specifies the range and height axis scale power.

Example: 0.5

Data Types

`double`

’SurfaceRefractivity’ - Surface refractivity

313 (default) | real-valued scalar

Surface refractivity, specified as a non-negative real-valued scalar. The surface refractivity is a parameter of the "CRPL Exponential Reference Atmosphere Model" on page 2-67 used in this function.

Example: 314

Data Types
`double`

'RefractionExponent' - Refraction exponent

0.143859 (default) | real-valued scalar

Refraction exponent specified as a non-negative, real-valued scalar. The refraction exponent is a parameter of the "CRPL Exponential Reference Atmosphere Model" on page 2-67 used in this function.

Example: 0.15

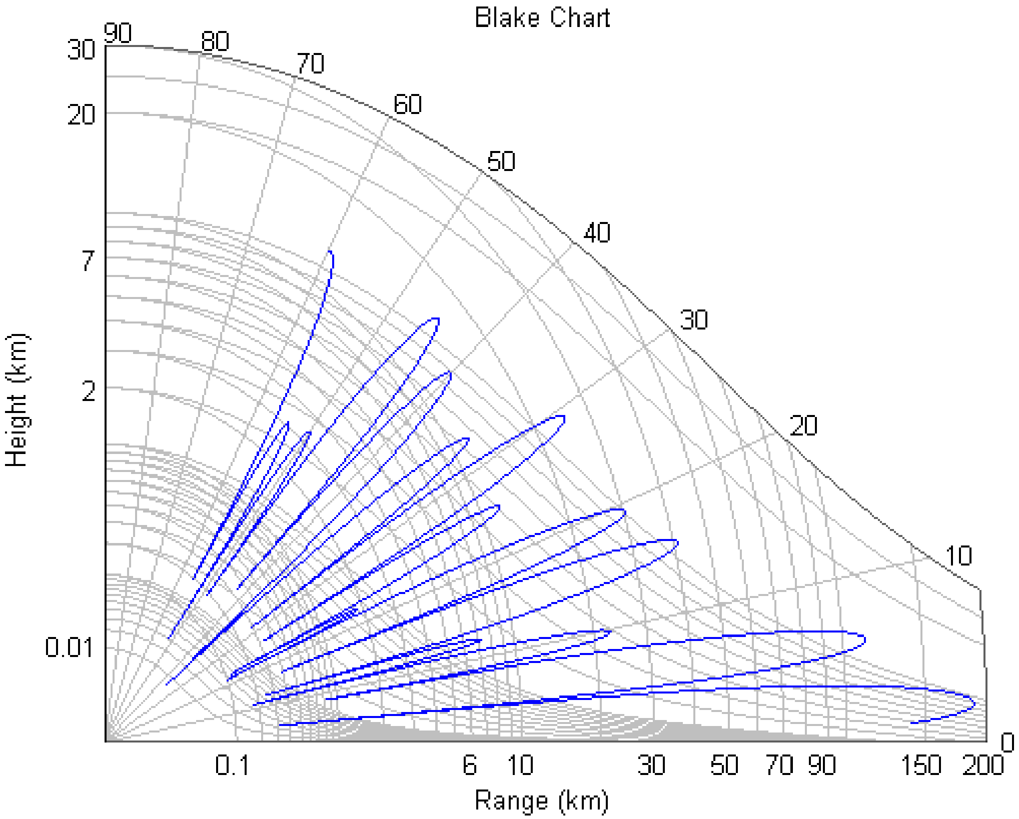
Data Types
`double`

Examples

Display Vertical Coverage Diagram

Display the vertical coverage diagram of an antenna transmitting at 100 MHz and placed 20 meters above the ground. Set the free-space range to 100 km. Use default plotting parameters.

```
freq = 100e6;
ant_height = 20;
rng_fs = 100;
[vcp, vcpangles] = radarvcd(freq,rng_fs,ant_height);
blakechart(vcp, vcpangles);
```

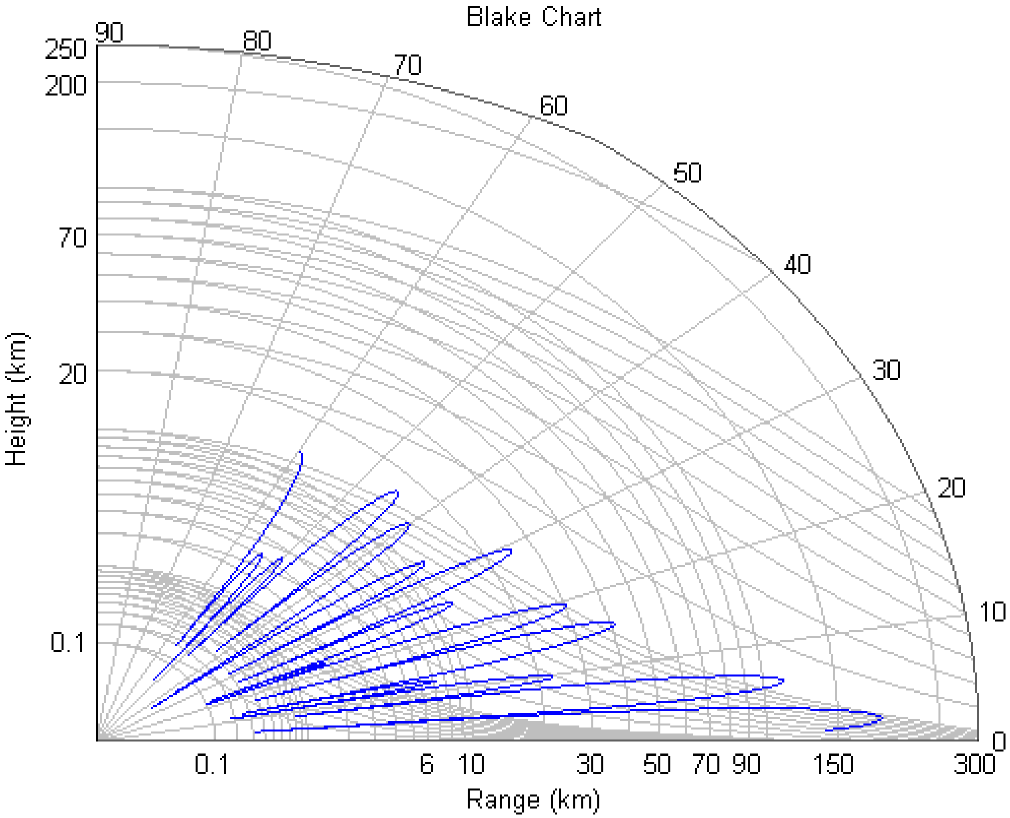


Display Vertical Coverage Diagram Specifying Maximum Range and Height

Display the vertical coverage diagram of an antenna transmitting at 100 MHz and placed 20 meters above the ground. Set the free-space range to 100 km. Set the maximum plotting range to 300 km and the maximum plotting height to 250 km.

```
freq = 100e6;
ant_height = 20;
```

```
rng_fs = 100;
[vcp, vcpangles] = radarvcd(freq,rng_fs,ant_height);
rmax = 300;
hmax = 250;
blakechart(vcp,vcpangles,rmax,hmax);
```



Display Vertical Coverage Diagram of Sinc Pattern Antenna

Plot the range-height-angle curve of a radar having a sinc-function antenna pattern.

Specify antenna pattern

Specify the antenna pattern as a sinc function.

```
pat_angles = linspace(-90,90,361)';
pat_u = 1.39157/sind(90/2)*sind(pat_angles);
pat = sinc(pat_u/pi);
```

Specify radar and environment parameters

Set the transmitting frequency to 100 MHz, the free-space range to 100 km, the antenna tilt angle to 0 degrees, and place the antenna 20 meters above the ground. Assume a surface roughness of one meter.

```
freq = 100e6;
ant_height = 10;
rng_fs = 100;
tilt_ang = 0;
surf_roughness = 1;
```

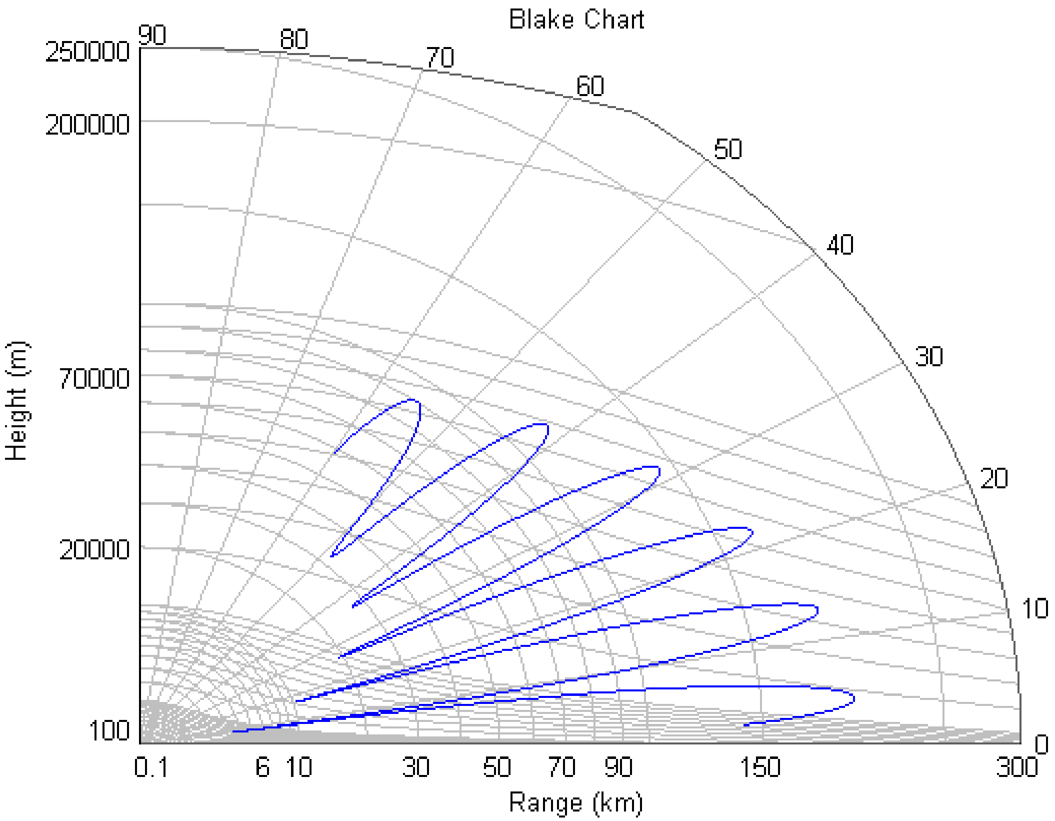
Create radar range-height-angle data

```
[vcp, vcpangles] = radarvcd(freq,rng_fs,ant_height,...
    'RangeUnit','km','HeightUnit','m',...
    'AntennaPattern',pat,...
    'PatternAngles',pat_angles,'TiltAngle',tilt_ang,...
    'SurfaceRoughness',surf_roughness);
```

Plot radar range-height-angle data

Set the maximum plotting range to 300 km and the maximum plotting height to 250,000 m. Choose the range units as kilometers, `'km'`, and the height units as meters, `'m'`. Set the range and height axes scale powers to 1/2.

```
rmax = 300;
hmax = 250e3;
blakechart(vcp, vcpangles, rmax, hmax, 'RangeUnit','km',...
    'ScalePower',1/2,'HeightUnit','m');
```



Definitions

CRPL Exponential Reference Atmosphere Model

The `blakechart` function uses the CRPL Exponential Reference Atmosphere to model refraction effects. The index of refraction is a function of height

$$n(h) = 1.0 + \left(N_s \times 10^{-6}\right) e^{-R_{exp} h}$$

where N_s is the atmospheric refractivity value (in units of 10^{-6}) at the surface of the earth, R_{exp} is a decay constant, and h is the height above the surface in kilometers. The default value of N_s is 313 and can be modified using the `'SurfaceRefractivity'` Name-Value pair. The default value of R_{exp} is 0.143859 and can be modified using the `'RefractionExponent'` Name-Value pair.

References

[1] Blake, L.V. *Machine Plotting of Radar Vertical-Plane Coverage Diagrams*. Naval Research Laboratory Report 7098, 1970.

See Also

`radarvcd`

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

Convert vector from Cartesian components to spherical representation

Syntax

`vs = cart2sphvec(vr,az,el)`

Description

`vs = cart2sphvec(vr,az,el)` converts the components of a vector or set of vectors, `vr`, from their representation in a local Cartesian coordinate system to a *spherical basis representation* contained in `vs`. A spherical basis representation is the set of components of a vector projected into a basis given by $(\mathbf{e}_{az},\mathbf{e}_{el},\mathbf{e}_{R})$. The orientation of a spherical basis depends upon its location on the sphere as determined by azimuth, `az`, and elevation, `el`.

Input Arguments

vr - Vector in Cartesian basis representation

3-by-1 column vector | 3-by-N matrix

Vector in Cartesian basis representation specified as a 3-by-1 column vector or 3-by-N matrix. Each column of `vr` contains the three components of a vector in the right-handed Cartesian basis *x,y,x*.

Example: `[4.0; -3.5; 6.3]`

Data Types
`double`
Complex Number Support: Yes

az - Azimuth angle

scalar in range [–180,180]

Azimuth angle specified as a scalar in the closed range [–180,180]. Angle units are in degrees. To define the azimuth angle of a point on a sphere, construct a vector from the origin to the point. The azimuth angle is the angle in the *xy*-plane from the positive *x*-axis to the vector's orthogonal projection into the *xy*-plane. As examples, zero azimuth angle and zero elevation angle specify a point on the *x*-axis while an azimuth angle of 90° and an elevation angle of zero specify a point on the *y*-axis.

Example: `45`

Data Types
`double`

el - Elevation angle

scalar in range [–90,90]

Elevation angle specified as a scalar in the closed range [–90,90]. Angle units are in degrees. To define the elevation of a point on the sphere, construct a vector from the origin to the point. The elevation angle is the angle from its orthogonal projection into the *xy*-plane to the vector itself. As examples, zero elevation angle defines the equator of the sphere and ±90° elevation define the north and south poles, respectively.

Example: `30`

Data Types
`double`

Output Arguments

vs - Vector in spherical basis

3-by-1 column vector | 3-by-N matrix

Spherical representation of a vector returned as a 3-by-1 column vector or 3-by-N matrix having the same dimensions as `vs`. Each column of `vs` contains the three components of the vector in the right-handed $(\mathbf{e}_{az},\mathbf{e}_{el},\mathbf{e}_R)$ basis.

Examples

Spherical Representation of Unit Z-Vector

Start with a vector in Cartesian coordinates pointing along the *z*-direction and located at 45° azimuth, 45° elevation. Compute its components with respect to the spherical basis at that point.

```
vr = [0;0;1];
vs = cart2sphvec(vr,45,45)
```

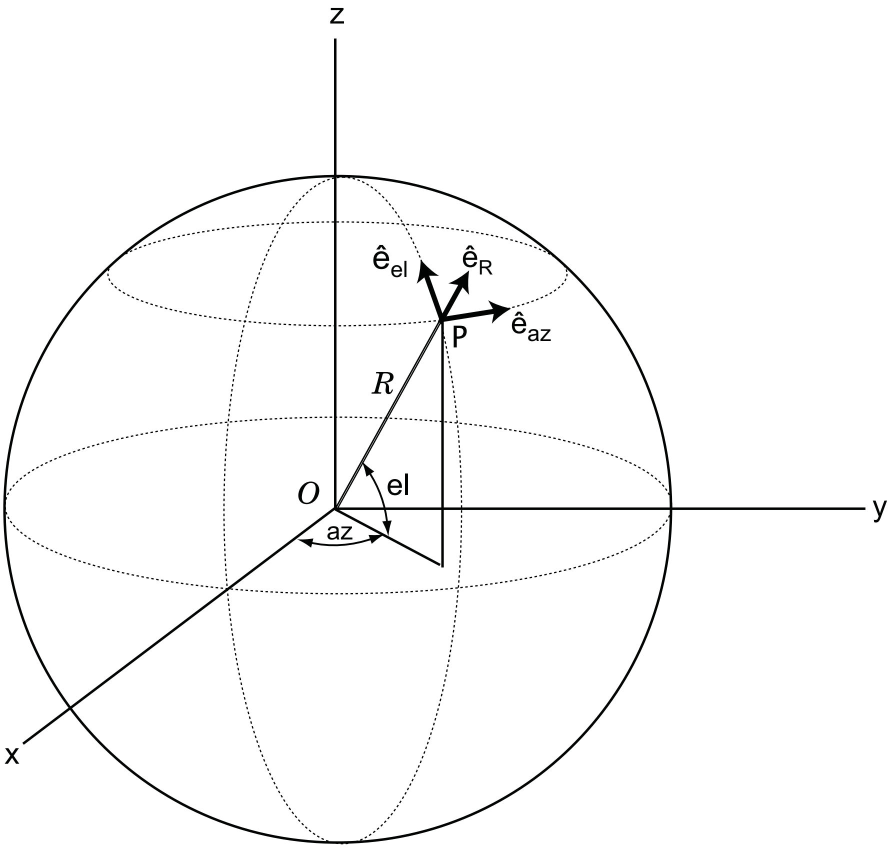
```
vs =

         0
    0.7071
    0.7071
```

Definitions

Spherical basis representation of vectors

The spherical basis is a set of three mutually orthogonal unit vectors $(\mathbf{e}_{az},\mathbf{e}_{el},\mathbf{e}_R)$ defined at a point on the sphere. The first unit vector points along lines of azimuth at constant radius and elevation. The second points along the lines of elevation at constant azimuth and radius. Both are tangent to the surface of the sphere. The third unit vector points radially outward.

The orientation of the basis changes from point to point on the sphere but is independent of R so as you move out along the radius, the basis orientation stays the same. The following figure illustrates the orientation of the spherical basis vectors as a function of azimuth and elevation:



For any point on the sphere specified by *az* and *el*, the basis vectors are given by:

$$
\begin{aligned}
\hat{\mathbf{e}}_{\mathbf{az}} &= -\sin(az)\hat{\mathbf{i}} + \cos(az)\hat{\mathbf{j}} \\
\hat{\mathbf{e}}_{\mathbf{el}} &= -\sin(el)\cos(az)\hat{\mathbf{i}} - \sin(el)\sin(az)\hat{\mathbf{j}} + \cos(el)\hat{\mathbf{k}} \\
\hat{\mathbf{e}}_{\mathbf{R}} &= \cos(el)\cos(az)\hat{\mathbf{i}} + \cos(el)\sin(az)\hat{\mathbf{j}} + \sin(el)\hat{\mathbf{k}} \quad .
\end{aligned}
$$

Any vector can be written in terms of components in this basis as $\mathbf{v} = v_{az}\hat{\mathbf{e}}_{\mathbf{az}} + v_{el}\hat{\mathbf{e}}_{\mathbf{el}} + v_R\hat{\mathbf{e}}_{\mathbf{R}}$. The transformations between spherical basis components and Cartesian components take the form

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} -\sin(az) & -\sin(el)\cos(az) & \cos(el)\cos(az) \\ \cos(az) & -\sin(el)\sin(az) & \cos(el)\sin(az) \\ 0 & \cos(el) & \sin(el) \end{bmatrix} \begin{bmatrix} v_{az} \\ v_{el} \\ v_R \end{bmatrix}$$

.

and

$$\begin{bmatrix} v_{az} \\ v_{el} \\ v_R \end{bmatrix} = \begin{bmatrix} -\sin(az) & \cos(az) & 0 \\ -\sin(el)\cos(az) & -\sin(el)\sin(az) & \cos(el) \\ \cos(el)\cos(az) & \cos(el)\sin(az) & \sin(el) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}.$$

See Also

`sph2cartvec` | `azelaxes`

cbfweights

Purpose

Conventional beamformer weights

Syntax

`wt = cbfweights(pos,ang)`

Description

`wt = cbfweights(pos,ang)` returns narrowband conventional beamformer weights. When applied to the elements of a sensor array, these weights steer the response of the array to a specified arrival direction or set of directions. The sensor array is defined by the sensor positions specified in the `pos` argument. The arrival directions are specified by azimuth and elevation angles in the `ang` argument. The output weights, `wt`, are returned as an *N*-by-*M* matrix. In this matrix, *N* represents the number of sensors in the array while *M* represents the number of arrival directions. Each column of `wt` contains the weights for the corresponding direction specified in the `ang`. The argument `wt` is equivalent to the output of the function `steervec` divided by *N*. All elements in the sensor array are assumed to be isotropic.

Input Arguments

pos - Positions of array sensor elements

1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. You define sensor position units in term of signal wavelength. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero. If `pos` is a 2-by-*N* matrix, then it represents the *(y,z)*-coordinates of the sensor elements of a planar array which is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. If `pos` is a 3-by-*N* matrix, then the array has arbitrary shape.

Example: [0, 0, 0; .1, .2, .3; 0,0,0]

Data Types
`double`

ang - Beamforming directions
1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Beamforming directions specified as a 1-by-*M* vector or a 2-by-*M* matrix. In this vector or matrix, *M* represents the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the beamforming direction as `[az;el]`. Angular units are specified in degrees. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the beamforming direction vector onto the *xy* plane. The angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the beamforming direction vector and *xy*-plane. It is positive when measured towards the positive *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero.

Example: [45;0]

Data Types
`double`

Output Arguments

wt - Beamformer weights
N-by-*M* complex-valued matrix

Beamformer weights returned as a *N*-by-*M* complex-valued matrix. In this matrix, *N* represents the number of sensor elements of the array while *M* represents the number of beamforming directions. Each column of `wt` corresponds to a beamforming direction specified in `ang`.

Examples

Weights for Two Beamformer Directions

Specify a line array of five elements spaced 10 cm apart. Compute the weights for two directions: 30° azimuth, 0° elevation, and 45° azimuth, 0° elevation. Assume a plane wave having a frequency of 1 GHz.

```
elementPos = (0:.1:.4); % meters
c = physconst('LightSpeed'); % speed of light;
fc = 1e9; % frequency
```

```
lam = c/fc; % wavelength
ang = [30 45]; % beamforming direction
wt = cbfweights(elementPos/lam,ang) % weights

wt =

   0.2000 + 0.0000i   0.2000 + 0.0000i
   0.0999 + 0.1733i   0.0177 + 0.1992i
  -0.1003 + 0.1731i  -0.1969 + 0.0353i
  -0.2000 - 0.0004i  -0.0527 - 0.1929i
  -0.0995 - 0.1735i   0.1875 - 0.0696i
```

References

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

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

See Also

`lcmvweights` **|** `mvdrweights` **|** `sensorcov` **|** `steervec` **|** `phased.PhaseShiftBeamformer`

Purpose

Convert circular component representation of field to linear component representation

Syntax

`fv = circpol2pol(cfv)`

Description

`fv = circpol2pol(cfv)` converts the circular polarization components of the field or fields contained in `cfv` to their linear polarization components contained in `fv`. Any polarized field can be expressed as a linear combination of horizontal and vertical components.

Input Arguments

cfv - Field vector in circular polarization representation

1-by-N complex-valued row vector or 2-by-N complex-valued matrix

Field vector in its circular polarization representation specified as a 1-by-N complex row vector or a 2-by-N complex matrix. If `cfv` is a matrix, each column represents a field in the form of `[El;Er]`, where `El` and `Er` are the left and right circular polarization components of the field. If `cfv` is a row vector, each column in `cfv` represents the polarization ratio, `Er/El`. For a row vector, the value `Inf` can designate the case when the ratio is computed for `El = 0`.

Example: [1;-1]

Data Types
`double`
Complex Number Support: Yes

Output Arguments

fv - Field vector in linear polarization representation or Jones vector

1-by-N complex-valued row vector or 2-by-N complex-valued matrix

Field vector in linear polarization representation or Jones vector returned as a 1-by-N complex-valued row vector or 2-by-N complex-valued matrix. `fv` has the same dimensions as `cfv`. If `cfv` is a matrix, each column of `fv` contains the horizontal and vertical linear polarization components of the field in the form, `[Eh;Ev]`. If `cfv` is a row vector, each entry in `fv` contains the linear polarization ratio, defined as `Ev/Eh`.

Examples

Linear Polarization Components from Circular Polarization Components

Convert a horizontally polarized field, originally expressed in circular polarization components, into linear polarization components.

```
cfv = [1;1];
fv = circpol2pol(cfv)
```

```
fv =

    1.4142
         0
```

The vertical component of the output is zero for horizontally polarized fields.

Linear Polarization Ratio from Circular Polarization Ratio

Create a right circularly polarized field. Compute the circular polarization ratio and convert to the linear polarization ratio equivalent. Note that the input circular polarization ratio is `Inf`.

```
cfv = [0;1];
q = cfv(2)/cfv(1);
p = circpol2pol(q)
```

```
p =

        0 - 1.0000i
```

References

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

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

See Also

`pol2circpol` | `polellip` | `polratio` | `stokes`

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.

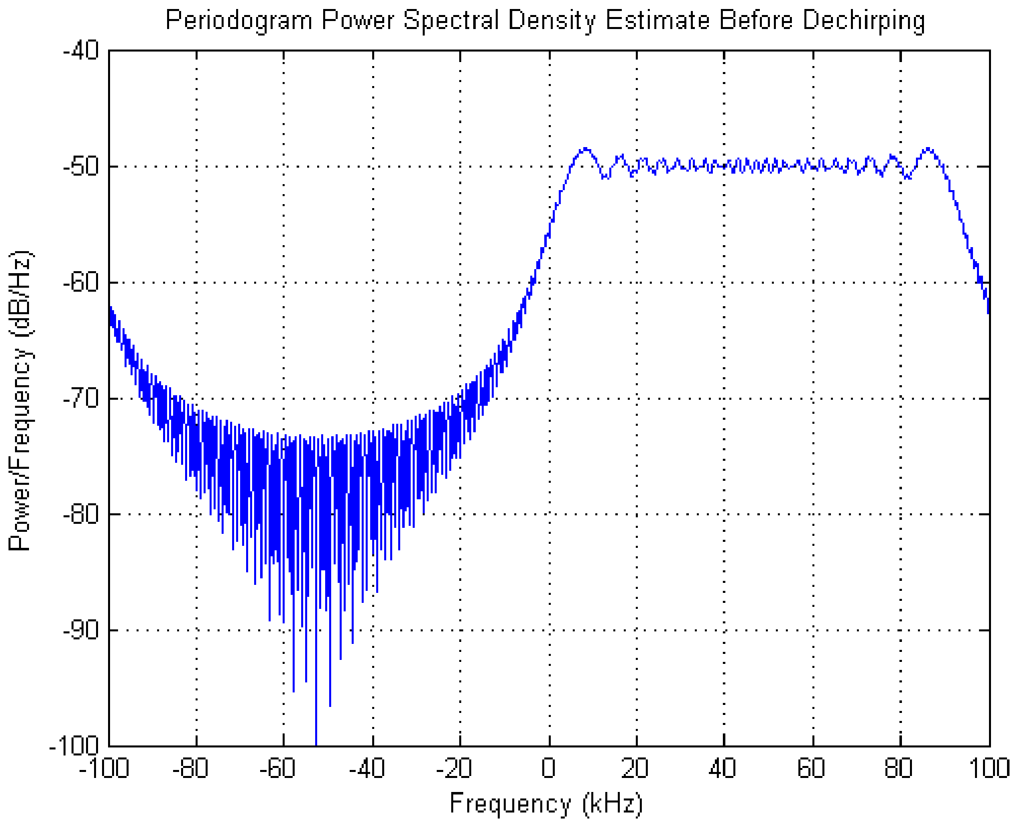
```
Fs = 2e5; Tm = 0.001;
hwav = phased.FMCWWaveform('SampleRate',Fs,'SweepTime',Tm);
xref = step(hwav);
```

Dechirp a delayed copy of the signal.

```
x = [zeros(10,1); xref(1:end-10)];
y = dechirp(x,xref);
```

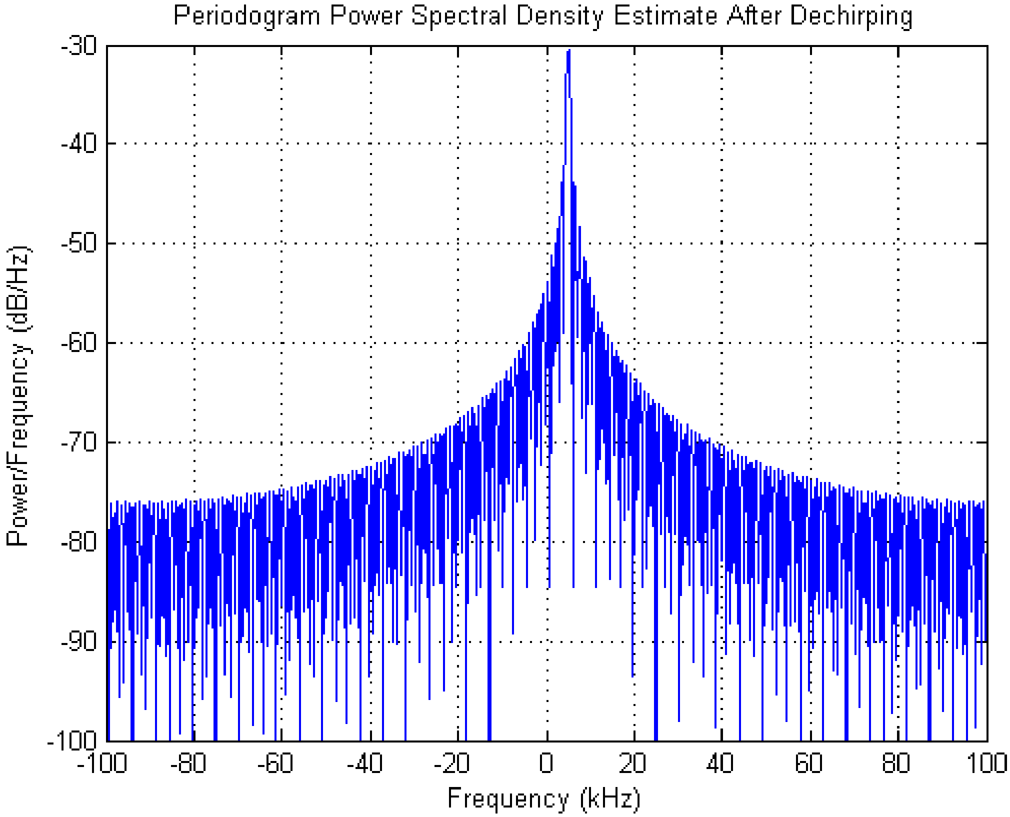
Plot the spectrum before dechirping.

```
[Pxx,F] = periodogram(x,[],1024,Fs,'centered');
plot(F/1000,10*log10(Pxx)); grid;
xlabel('Frequency (kHz)');
ylabel('Power/Frequency (dB/Hz)');
title('Periodogram Power Spectral Density Estimate Before Dechirping')
```



Plot the spectrum after dechirping.

```
[Pyy,F] = periodogram(y,[],1024,Fs,'centered');
plot(F/1000,10*log10(Pyy));
xlabel('Frequency (kHz)');
ylabel('Power/Frequency (dB/Hz)');
ylim([-100 -30]); grid
title('Periodogram Power Spectral Density Estimate After Dechirping');
```



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');
```

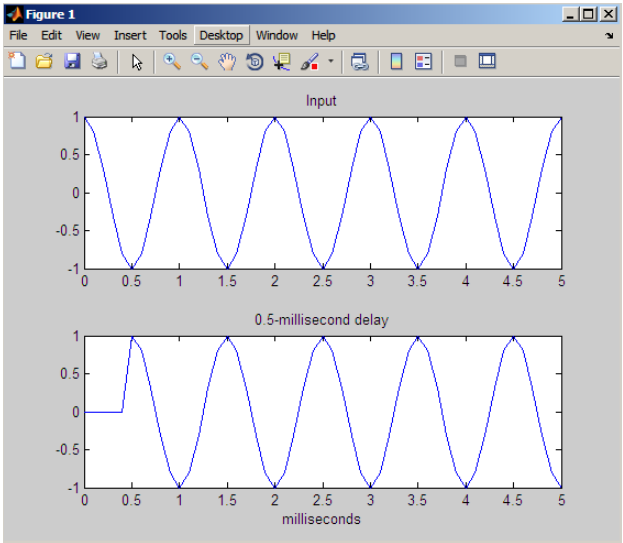


Figure 1
File Edit View Insert Tools Desktop Window Help
Input
1
0.5
0
-0.5
-1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
0.5-millisecond delay
1
0.5
0
-0.5
-1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
milliseconds

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');
```

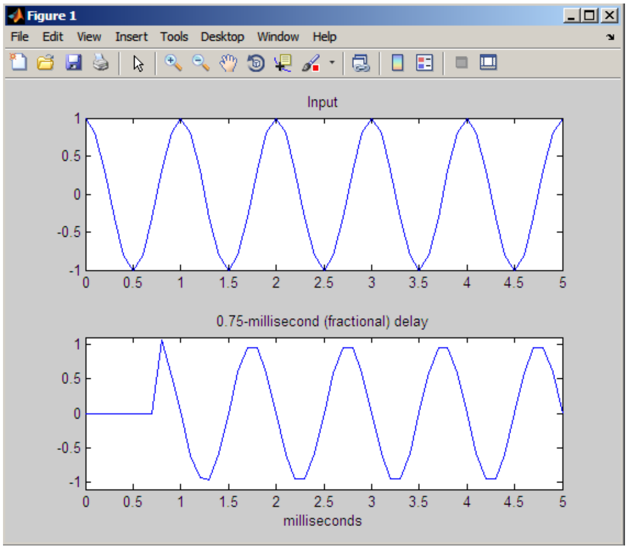


Figure 1
File Edit View Insert Tools Desktop Window Help
Input
1
0.5
0
-0.5
-1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
0.75-millisecond (fractional) delay
1
0.5
0
-0.5
-1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
milliseconds

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`

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

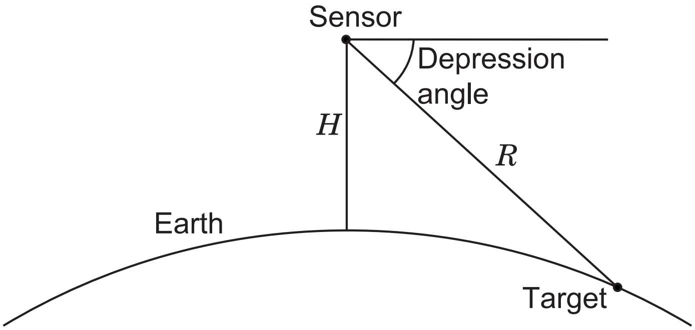
depAng

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`

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:

$$\frac{1}{1+r\cdot\texttt{RGradient}}$$

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 $-39 \times 10^{-9}\ \text{m}^{-1}$.

References

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

See Also

`depressionang` | `horizonrange`

Purpose

Direction of arrival using TLS ESPRIT

Syntax

`ang = espritdoa(R,nsig)`
`ang = espritdoa( ___ ,Name,Value)`

Description

`ang = espritdoa(R,nsig)` estimates the directions of arrival, `ang`, of a set of plane waves received on a uniform line array (ULA). The estimation employs the *TLS ESPRIT*, the total least-squares ESPRIT, algorithm. The input arguments are the estimated spatial covariance matrix between sensor elements, `R`, and the number of arriving signals, `nsig`. In this syntax, sensor elements are spaced one-half wavelength apart.

`ang = espritdoa( ___ ,Name,Value)` estimates the directions of arrival with additional options specified by one or more `Name,Value` pair arguments. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

R - Spatial covariance matrix

Complex-valued positive-definite N-by-N matrix.

Spatial covariance matrix, specified as a complex-valued, positive-definite, N-by-N matrix. In this matrix, N represents the number of elements in the ULA array. If `R` is not Hermitian, a Hermitian matrix is formed by averaging the matrix and its conjugate transpose, `(R+R')/2`.

Example: [4.3162, –0.2777 – 0.2337i; –0.2777 + 0.2337i , 4.3162]

Data Types
`double`
Complex Number Support: Yes

nsig - Number of arriving signals

Positive integer

Number of arriving signals, specified as a positive integer.

Example: 3

Data Types
`double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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`.

Example: 'ElementSpacing', 0.45

'ElementSpacing' - ULA element spacing

0.5 (default) | Real-valued positive scalar

ULA element spacing, specified as a real-valued, positive scalar. Position units are measured in terms of signal wavelength.

Example: 0.4

Data Types
`double`

'RowWeighting' - Row weights

1 (default) | Real-valued positive scalar

Row weights specified as a real-valued positive scalar. These weights are applied to the selection matrices which determine the ESPRIT subarrays. A larger value is generally better but the value must be less than or equal to $(N_s–1)/2$, where N_s is the number of subarray elements. The number of subarray elements is $N_s = N–1$. The value of N is the number of ULA elements, as specified by the dimensions of the spatial covariance matrix, `R`. A detailed discussion of selection matrices and row weighting can be found in Van Trees [1], p. 1178.

Example: 5

Data Types
`double`

Output Arguments

ang - Directions of arrival angles

Real-valued 1-by-*M* row vector

Directions of arrival angle returned as a real-valued, 1-by-*M* vector. The dimension *M* is the number of arriving signals specified in the argument, `nsig`. This angle is the broadside angle. Angle units are degrees and angle values lie between –90° and 90°.

Examples

Three Signals Arriving at Half-Wavelength-Spaced ULA

Assume a half-wavelength spaced uniform line array with 10 elements. Three plane waves arrive from the 0°, –25°, and 30° azimuth directions. Elevation angles are 0°. The noise is spatially and temporally white. The SNR for each signal is 5 dB. Find the arrival angles.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25 30];
Nsig = 3;
R = sensorcov(elementPos,angles,db2pow(-5));
doa = espritdoa(R,Nsig)
```

```
doa =

   30.0000    0.0000  -25.0000
```

The `espritdoa` functions produces the correct angles.

Three Signals Arriving at 0.4-Wavelength-Spaced ULA

Assume a uniform line array with 10 element. The element spacing is smaller than one-half wavelength. Three plane waves arrive from the 0°, –25°, and 30° azimuth directions. Elevation angles are 0°. The noise is spatially and temporally white. The SNR for each signal is 5 dB.

Set the `ElementSpacing` property value to the interelement spacing. Find the arrival angles.

```
N = 10;
```

```
d = 0.4;
elementPos = (0:N-1)*d;
angles = [0 -25 30];
Nsig = 3;
R = sensorcov(elementPos,angles,db2pow(-5));
doa = espritdoa(R,Nsig,'ElementSpacing',d)

doa =

   30.0000    0.0000  -25.0000
```

The `espritdoa` functions again produces the correct angles.

References

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

See Also

`aictest` **|** `mdltest` **|** `rootmusicdoa` **|** `spsmooth` **|** `phased.ESPRITEstimator`

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`

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.

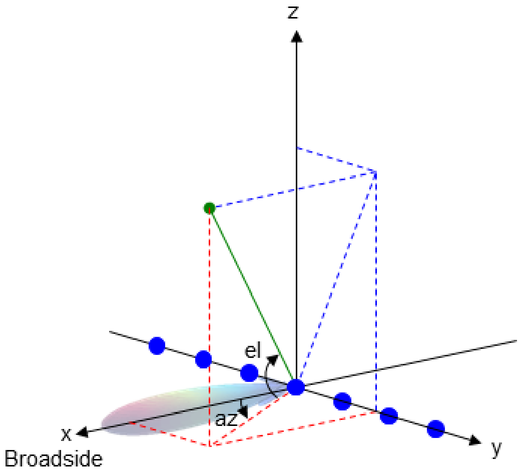
Definitions

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"

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

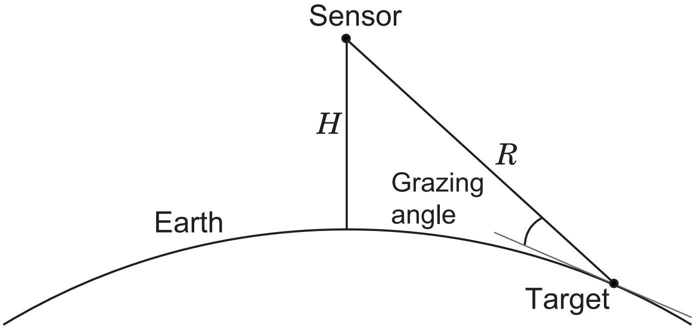
grazAng

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`

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

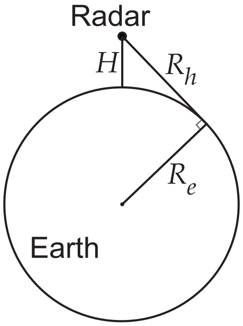
Rh

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`

lcmvweights

Purpose

Narrowband linearly constrained minimum variance (LCMV) beamformer weights

Syntax

`wt = lcmvweights(constr,resp,cov)`

Description

`wt = lcmvweights(constr,resp,cov)` returns narrowband linearly-constrained minimum variance (LCMV) beamformer weights, `wt`, for a phased array. When applied to the elements of the array, these weights steer the response of the array toward a specific arrival direction or set of directions. LCMV beamforming requires that the beamformer response to signals from a direction of interest are passed with specified gain and phase delay. However, power from interfering signals and noise from all other directions is minimized. Additional constraints may be imposed to specifically nullify output power coming from known directions. The constraints are contained in the matrix, `constr`. Each column of `constr` represents a separate constraint vector. The desired response to each constraint is contained in the response vector, `resp`. The argument `cov` is the sensor spatial covariance matrix. All elements in the sensor array are assumed to be isotropic.

Input Arguments

constr - Constraint matrix

N-by-K complex-valued matrix

Constraint matrix specified as a complex-valued, N-by-K, complex-valued matrix. In this matrix N represents the number of elements in the sensor array while K represents the number of constraints. Each column of the matrix specifies a constraint on the beamformer weights. The number of K must be less than or equal to N.

Example: [0, 0, 0; .1, .2, .3; 0,0,0]

Data Types
`double`
Complex Number Support: Yes

resp - Desired response

K-by-1 complex-valued column vector.

Desired response specified as complex-valued, K-by-1 column vector where K is the number of constraints. The value of each element in the vector is the desired response to the constraint specified in the corresponding column of `constr`.

Example: [45;0]

Data Types
`double`
Complex Number Support: Yes

cov - Sensor spatial covariance matrix

N-by-N complex-valued matrix

Sensor spatial covariance matrix specified as a complex-valued, N-by-N matrix. In this matrix, N represents the number of sensor elements. The covariance matrix consists of the variances of the element data and the covariance between sensor elements. It contains contributions from all incoming signals and noise.

Example: [45;0]

Data Types
`double`
Complex Number Support: Yes

Output Arguments

wt - Beamformer weights

N-by-1 complex-valued vector

Beamformer weights returned as an N-by-1, complex-valued vector. In this vector, N represents the number of elements in the array.

Examples

LCMV Beamformer with Nulls at -40 and 20 degrees

Construct a 10-element half-wavelength-spaced line array. Then, compute the LCMV weights for a desired arrival direction of 0 degrees azimuth. Impose three direction constraints : a null at -40 degrees, a unit desired response in the arrival direction 0 degrees, and another null at 20 degrees. The sensor spatial covariance matrix includes two signals arriving from -60 and 60 degrees and -10 dB isotropic white noise.

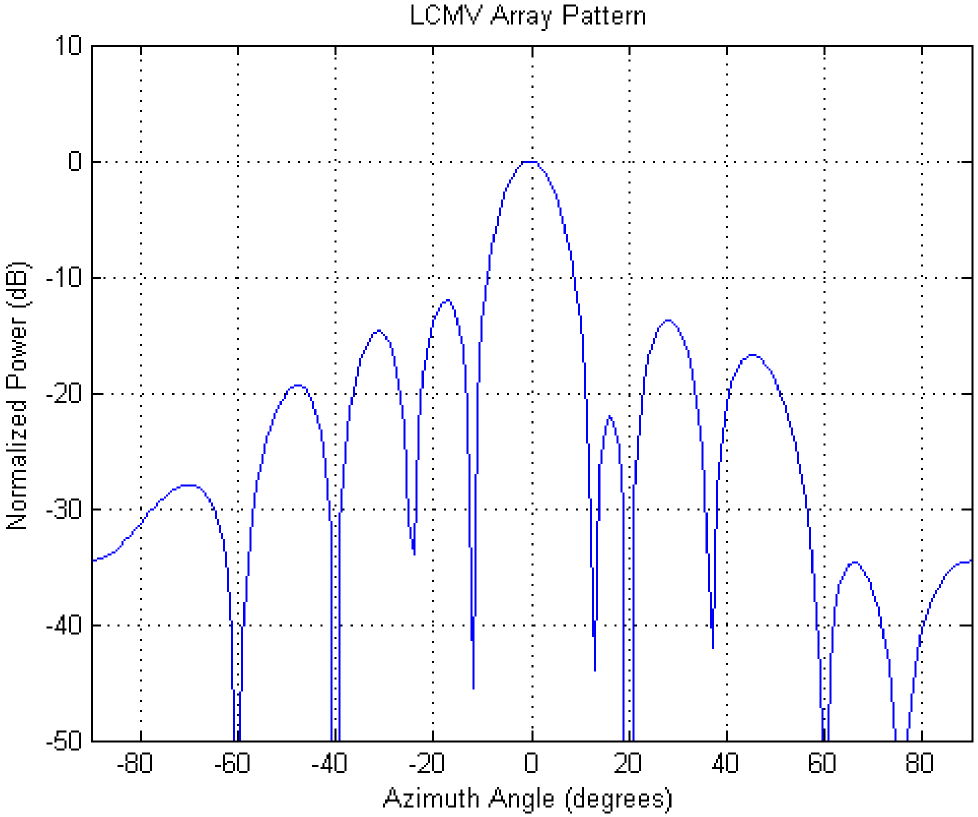
```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
sv = steervec(elementPos,[-40 0 20]);
resp = [0 1 0]';
Sn  = sensorcov(elementPos,[-60 60],db2pow(-10));
```

Compute the beamformer weights.

```
w = lcmvweights(sv,resp,Sn);
```

Plot the array pattern for the computed weights.

```
vv = steervec(elementPos,[-90:90]);
plot([-90:90],mag2db(abs(w'*vv)))
grid on
axis([-90,90,-50,10]);
xlabel('Azimuth Angle (degrees)');
ylabel('Normalized Power (dB)');
title('LCMV Array Pattern');
```



The above figure shows that maximum gain is attained at 0 degrees as expected. In addition, the constraints impose nulls at -40 and 20 degrees and these can be seen in the plot. The nulls at -60 and 60 degrees arise from the fundamental property of the LCMV beamformer of suppressing the power contained in the two plane waves that contributed to the sensor spatial covariance matrix.

Definitions

Linear-Constrained Minimum Variance Beamformers

The LCMV beamformer computes weights that minimize the total output power of an array but that are subject to some constraints (see Van Trees [1], p. 527). In order to steer the response of the array to a particular arrival direction, weights are chosen to produce unit gain when applied to the steering vector for that direction. This requirement can be thought of as a constraint on the weights. Additional constraints may be applied to nullify the array response to signals from other arrival directions such as those containing noise sources. Let $(az_1,el_1),(az_2,el_2),\ldots,(az_K,el_K)$ be the set of directions for which a constraint is to be imposed. Each direction has a corresponding steering vector, $\mathbf{c}_k$, and the response of the array to that steering vector is given by $\mathbf{c}_k^H\mathbf{w}$. The transpose conjugate of a vector is denoted by the superscript symbol H. A constraint is imposed when a desired response is required when the beamformer weights act on a steering vector, $\mathbf{c}_k$,

$$\mathbf{c}_k^H\mathbf{w} = r_k$$

This response could be specified as unity to allow the array to pass through the signal from a certain direction. It could be zero to nullify the response from that direction. All the constraints can be collected into a single matrix, C, and all the response into a single column vector, $\mathbf{R}$. This allows the constraints to be represented together in matrix form

$$C^H\mathbf{w} = \mathbf{R}$$

The LCMV beamformer chooses weights to minimize the total output power

$$P = \mathbf{w}^H S\mathbf{w}$$

subject to the above constraints. S denotes the sensor spatial covariance matrix. The solution to the power minimization is

$$\mathbf{w} = S^{-1}C^H\left(CS^{-1}C^H\right)^{-1}\mathbf{R}$$

and its derivation can be found in [2].

References

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

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. "Beamforming: A versatile approach to spatial filtering". *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

See Also

`cbfweights` **|** `mvdrweights` **|** `sensorcov` **|** `steervec` **|** `phased.LCMVBeamformer`

local2globalcoord

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

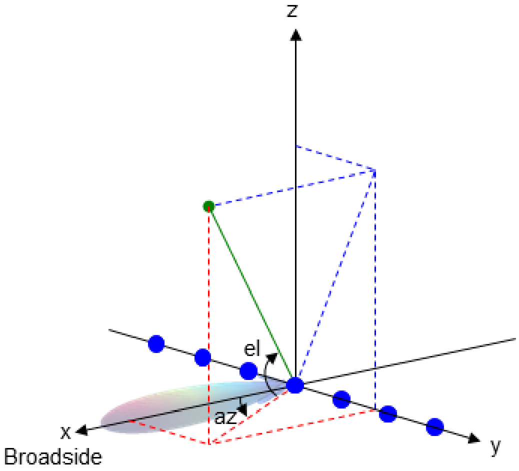
Azimuth Angle, Elevation Angle

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"

mdltest

Purpose

Dimension of signal subspace

Syntax

```
nsig = mdltest(X)
nsig = mdltest(X,'fb')
```

Description

`nsig = mdltest(X)` estimates the number of signals, `nsig`, present in a *snapshot* of data, `X`, that impinges upon the sensors in an array. The estimator uses the *Minimum Discription Length* (MDL) test. The input argument, `X`, is a complex-valued matrix containing a time sequence of data samples for each sensor. Each row corresponds to a single time sample for all sensors.

`nsig = mdltest(X,'fb')` estimates the number of signals. Before estimating, it performs *forward-backward averaging* on the sample covariance matrix constructed from the data snapshot, `X`. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

X - Data snapshot

Complex-valued K-by-N matrix

Data snapshot, specified as a complex-valued, K-by-N matrix. A snapshot is a sequence of time-samples taken simultaneous at each sensor. In this matrix, K represents the number of time samples of the data, while N represents the number of sensor elements.

Example: [–0.1211 + 1.2549i, 0.1415 + 1.6114i, 0.8932 + 0.9765i;]

Data Types
`double`
Complex Number Support: Yes

Output Arguments

nsig - Dimension of signal subspace

Non-negative integer

Dimension of signal subspace, returned as a non-negative integer. The dimension of the signal subspace is the number of signals in the data.

Examples

Estimate the Signal Subspace Dimensions for Two Arriving Signals

Construct a data snapshot for two plane waves arriving at a half-wavelength-spaced uniform line array with 10 elements. The plane waves arrive from 0° and –25° azimuth, both with elevation angles of 0°. Assume the signals arrive in the presence of additive noise that is both temporally and spatially Gaussian white noise. For each signal, the SNR is 5 dB. Take 300 samples to build a 300-by-10 data snapshot. Then, solve for the number of signals using `mdltest`.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25];
x = sensorsig(elementPos,300,angles,db2pow(-5));
Nsig = mdltest(x)

Nsig =

     2
```

The result shows that the number of signals is two, as expected.

Estimate the Signal Subspace Dimensions Using Forward-Backward Averaging

Construct a data snapshot for two plane waves arriving at a half-wavelength-spaced uniform line array with 10 elements. Correlated plane waves arrive from 0° and 10° azimuth, both with elevation angles of 0°. Assume the signals arrive in the presence of additive noise that is both temporally and spatially Gaussian white noise. For each signal, the SNR is 10 dB. Take 300 samples to build a 300-by-10 data snapshot. Then, solve for the number of signals using `mdltest`.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
```

```
angles = [0 10];
ncov = db2pow(-10);
scov = [1 .5]'*[1 .5];
x = sensorsig(elementPos,300,angles,ncov,scov);
Nsig = mdltest(x)

Nsig =

     1
```

This result shows that `aictest` function cannot determine the number of signals correctly when the signals are correlated.

Now, try the option of forward-backward smoothing.

```
Nsig = mdltest(x,'fb')

Nsig =

     2
```

The addition of forward-backward smoothing yields the correct number of signals.

Definitions

Estimating the Number of Sources

AIC and MDL tests

Direction finding algorithms such as MUSIC and ESPRIT require knowledge of the number of sources of signals impinging on the array or equivalently, the dimension, d, of the signal subspace. The Akaike Information Criterion (AIC) and the Minimum Description Length (MDL) formulas are two frequently-used estimators for obtaining that dimension. Both estimators assume that, besides the signals, the data contains spatially and temporally white Gaussian random noise. Finding the number of sources is equivalent to finding the multiplicity of the smallest eigenvalues of the sampled spatial covariance matrix. The sample spatial covariance matrix constructed from a data snapshot is used in place of the actual covariance matrix.

A requirement for both estimators is that the dimension of the signal subspace be less than the number of sensors, *N*, and that the number of time samples in the snapshot, *K*, be much greater than *N*.

A variant of each estimator exists when forward-backward averaging is employed to construct the spatial covariance matrix. Forward-backward averaging is useful for the case when some of the sources are highly correlated with each other. In that case, the spatial covariance matrix may be ill conditioned. Forward-backward averaging can only be used for certain types of symmetric arrays, called *centro-symmetric* arrays. Then the forward-backward covariance matrix can be constructed from the sample spatial covariance matrix, *S*, using $S_{FB} = S + JS^*J$ where *J* is the exchange matrix. The exchange matrix maps array elements into their symmetric counterparts. For a line array, it would be the identity matrix flipped from left to right.

All the estimators are based on a cost function

$$L_d(d) = K(N-d)\ln\left\{\frac{\dfrac{1}{N-d}\displaystyle\sum_{i=d+1}^{N}\lambda_i}{\left\{\displaystyle\prod_{i=d+1}^{N}\lambda_i\right\}^{\frac{1}{N-d}}}\right\}$$

plus an added penalty term. The value λ_i represent the smallest *(N–d)* eigenvalues of the spatial covariance matrix. For each specific estimator, the solution for *d* is given by

- AIC

$$\hat{d}_{AIC} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + d(2N-d)\right\}$$

- AIC for forward-backward averaged covariance matrices

$$\hat{d}_{AIC:FB} = \underset{d}{\operatorname{argmin}}\left\{L_d(d) + \frac{1}{2}d(2N-d+1)\right\}$$

- MDL

$$\hat{d}_{MDL} = \underset{d}{\operatorname{argmin}} \left\{ L_d(d) + \frac{1}{2}(d(2N-d)+1)\ln K \right\}$$

- MDL for forward-backward averaged covariance matrices

$$\hat{d}_{MDL\,FB} = \underset{d}{\operatorname{argmin}} \left\{ L_d(d) + \frac{1}{4} d(2N-d+1)\ln K \right\}$$

References

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

See Also

`aictest` **|** `espritdoa` **|** `rootmusicdoa` **|** `spsmooth`

mvdrweights

Purpose

Minimum variance distortionless response (MVDR) beamformer weights

Syntax

`wt = mvdrweights(pos,ang,cov)`

Description

`wt = mvdrweights(pos,ang,cov)` returns narrowband minimum variance distortionless response (MVDR) beamformer weights for a phased array. When applied to the elements of an array, they steer the response of a sensor array in a specific arrival direction or set of directions. The sensor array is defined by the sensor positions specified in the `pos` argument. The arrival directions are specified by azimuth and elevation angles in the `ang` argument. `cov` is the sensor spatial covariance matrix between sensor elements. The output argument, `wt`, is a matrix contains the beamformer weights for each sensor and each direction. Each column of `wt` contains the weights for the corresponding direction specified in `ang`. All elements in the sensor array are assumed to be isotropic.

Input Arguments

pos - Positions of array sensor elements

1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. You define sensor position units in term of signal wavelength. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero. If `pos` is a 2-by-*N* matrix, then it represents the *(y,z)*-coordinates of the sensor elements of a planar array which is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. If `pos` is a 3-by-*N* matrix, then the array has arbitrary shape.

Example: [0, 0, 0; .1, .2, .3; 0,0,0]

Data Types

`double`

ang - Beamforming directions

1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Beamforming directions specified as a 1-by-*M* vector or a 2-by-*M* matrix. In this vector or matrix, *M* represents the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the beamforming direction as `[az;el]`. Angular units are specified in degrees. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the beamforming direction vector onto the *xy* plane. The angle is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the beamforming direction vector and *xy*-plane. It is positive when measured towards the positive *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero.

Example: [45;0]

Data Types
`double`

cov - Sensor spatial covariance matrix

N-by-*N* complex-valued matrix

Sensor spatial covariance matrix specified as an *N*-by-*N*, complex-valued matrix. In this matrix, *N* represents the number of sensor elements. The covariance matrix consists of the variances of the element data and the covariances of the data between the sensor elements and contains contributions from all incoming signals and noise.

Example: [45;0]

Data Types
`double`
Complex Number Support: Yes

Output Arguments

wt - Beamformer weights

N-by-M complex-valued matrix

Beamformer weights returned as a complex-valued, N-by-M matrix. In this matrix, N represents the number of sensor elements of the array while M represents the number of beamforming directions. Each column of `wt` corresponds to a beamforming direction specified in `ang`.

Examples

MVDR Beamformer with Arrival Directions of 30 and 45 Degrees

Construct a 10-element, half-wavelength-spaced line array. Choose two arrival directions of interest - one at 30 degrees azimuth and the other at 45 degrees azimuth. Assume both directions have 0 degrees elevation. Compute the MVDR beamformer weights for each direction. Specify a sensor spatial covariance matrix that contains signals arriving from -60 and 60 degrees and noise at -10 dB.

Set up the array and sensor spatial covariance matrix.

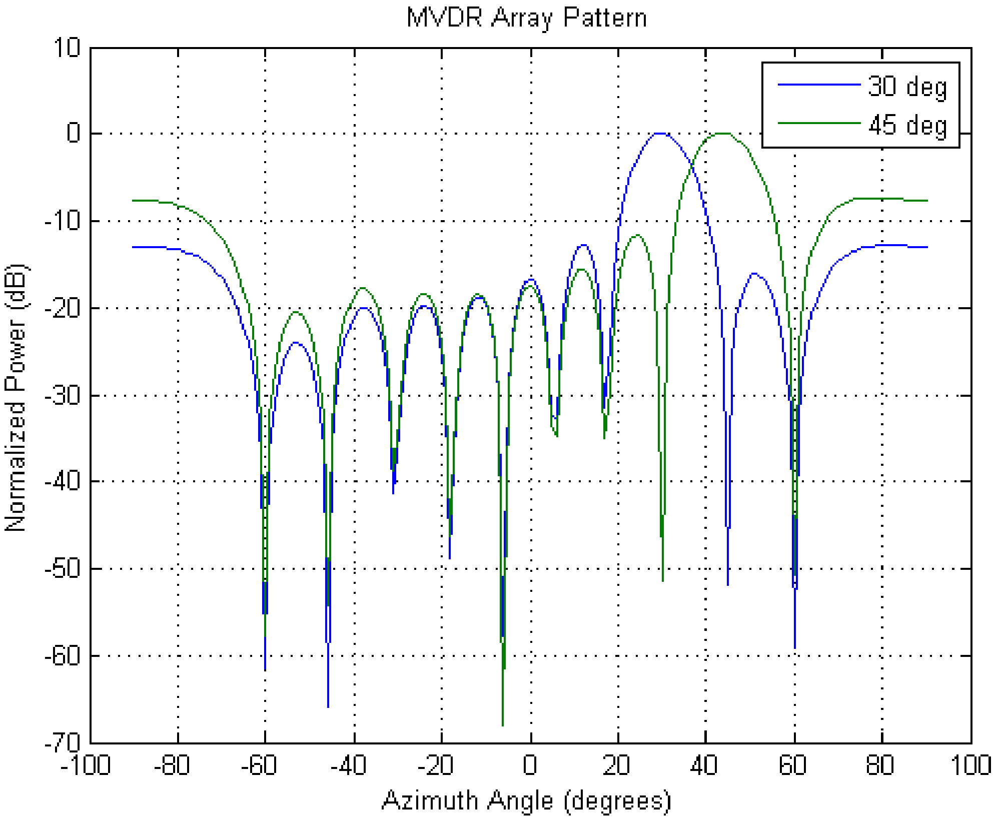
```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
Sn  = sensorcov(elementPos,[-60 60],db2pow(-10));
```

Solve for the MVDR beamformer weights.

```
w = mvdrweights(elementPos,[30 45],Sn);
```

Plot the two MVDR array patterns.

```
plotangl = -90:90;
vv = steervec(elementPos,plotangl);
plot(plotangl,mag2db(abs(w'*vv)))
grid on
xlabel('Azimuth Angle (degrees)');
ylabel('Normalized Power (dB)');
legend('30 deg','45 deg');
title('MVDR Array Pattern')
```



The figure shows plots for each beamformer direction. One plot has the expected maximum gain at 30 degrees and the other at 45 degrees. The nulls at -60 and 60 degrees arise from the fundamental property of the MVDR beamformer of suppressing power in all directions except for the arrival direction.

Definitions

Minimum variance distortionless response

The MVDR beamformer computes weights that minimize the total output power of an array but sets the gain in one particular direction to unity (see Van Trees [1], p. 442). If the steering vector, $\mathbf{v}_0$, corresponds to the direction of interest, then the MVDR weights are given by

$$\mathbf{w} = \frac{S^{-1|}\mathbf{v_0}}{\mathbf{v_0}^H S^{-1}\mathbf{v_0}}$$

where S is the spatial covariance matrix.

References

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

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

See Also

`steervec` | `cbfweights` | `lcmvweights` | `sensorcov` | `phased.MVDRBeamformer`

noisepow

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`

npwgnthresh

Purpose

Detection SNR threshold for signal in white Gaussian noise

Syntax

```
SNRTHRESH = npwgnthresh(PFA)
SNRTHRESH = npwgnthresh(PFA,NPULS)
SNRTHRESH = npwgnthresh(PFA,NPULS,DTYPE)
SNRTHRESH = npwgnthresh(PFA,NPULS,DTYPE,OUTSCALE)
```

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.

`SNRTHRESH = npwgnthresh(PFA,NPULS,DTYPE,OUTSCALE)` specifies the output scale, `OUTSCALE`, as 'db' or 'linear'.

Input Arguments

PFA

Probability of false alarm.

NPULS

Number of pulses used in the integration.

Default: `1`

DTYPE

Detection type.

Specifies 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'`

OUTSCALE

Output scale.

Specifies the scale of the output value. Valid choices for `OUTSCALE` are`'db'` or `'linear'`. When `OUTSCALE` is set to `'linear'`, the returned threshold represents amplitude.

Default: `'db'`

Output Arguments

SNRTHRESH

Detection threshold expressed in signal-to-noise ratio in decibels or linear if `OUTSCALE` is set to`` `linear'``. The relationship between the linear threshold and the threshold in dB is

$$T_{dB} = 20\log_{10} T_{lin}$$

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 linear detection threshold λ for an NP detector can be expressed as

$$\frac{\lambda}{\sigma} = \sqrt{2N}\,\mathrm{erfc}^{-1}(2P_{fa})$$

The threshold can also be expressed as a signal-to-noise ratio in decibels:

$$10\log_{10}\left(\frac{\lambda^2}{\sigma^2}\right) = 10\log_{10}\left(2N\left(\mathrm{erfc}^{-1}(2P_{fa})\right)^2\right)$$

In these equations:

- σ^2 is the variance of the white Gaussian noise sequence
- N is the number of samples
- $erfc^{-1}$ is the inverse of the complementary error function
- P_{fa} is the probability of false alarm

Note For probabilities of false alarm greater than or equal to 1/2, the formula for detection threshold as SNR is invalid since erfc^{-1} is less than or equal to zero for values of its argument greater than or equal to one. In that case, use the linear output of the function invoked by setting `OUTSCALE` to `'linear'`.

Detection in Complex-Valued White Gaussian Noise (Coherent Samples)

The NP detector for complex-valued signals is similar to that discussed in "Detection in Real-Valued White Gaussian Noise" on page 2-139. In addition, the function makes these assumptions:

- The variance of the complex-valued Gaussian random variable is divided equally among the real and imaginary parts.
- The real and imaginary parts are uncorrelated.

Under these assumptions, the linear detection threshold for an NP detector is

$$\frac{\lambda}{\sigma} = \sqrt{N}\,\mathrm{erfc}^{-1}(2P_{fa})$$

and expressed as a signal-to-noise ratio in decibels is:

$$10\log_{10}\left(\frac{\lambda^2}{\sigma^2}\right) = 10\log_{10}\left(N\left(\mathrm{erfc}^{-1}(2P_{fa})\right)^2\right)$$

Note For probabilities of false alarm greater than or equal to 1/2, the formula for detection threshold as SNR is invalid since erfc^{-1} is less than or equal to zero for values of its argument greater than or equal to one. In that case, use the linear output of the function invoked by setting `OUTSCALE` to `'linear'`.

Detection of Noncoherent Samples in White Gaussian Noise

For noncoherent samples in white Gaussian noise, detection of a nonzero mean leads to a square-law detector. For a detailed derivation, see [2], pp. 324–329.

The linear detection threshold for the noncoherent NP detector is:

$$\frac{\lambda}{\sigma} = \sqrt{P^{-1}(N, 1 - P_{fa})}$$

The threshold expressed as a signal-to-noise ratio in decibels is:

$$10\log_{10}\left(\frac{\lambda^2}{\sigma^2}\right) = 10\log_{10} P^{-1}(N, 1 - P_{fa})$$

where $P^{-1}(x, y)$ is the inverse of the lower incomplete gamma function, P_{fa} is the probability of false alarm, and N is the number of pulses.

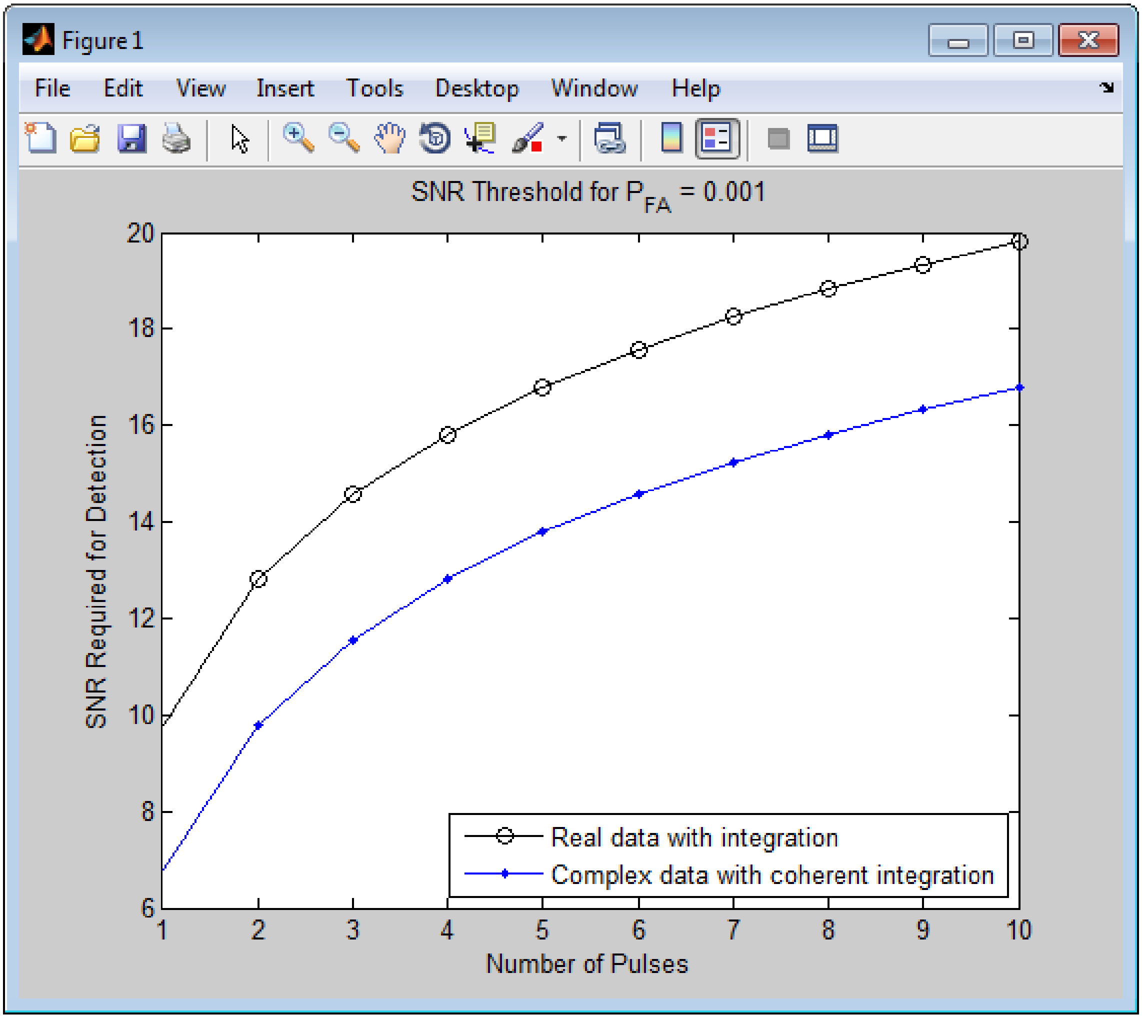
Examples

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;
```

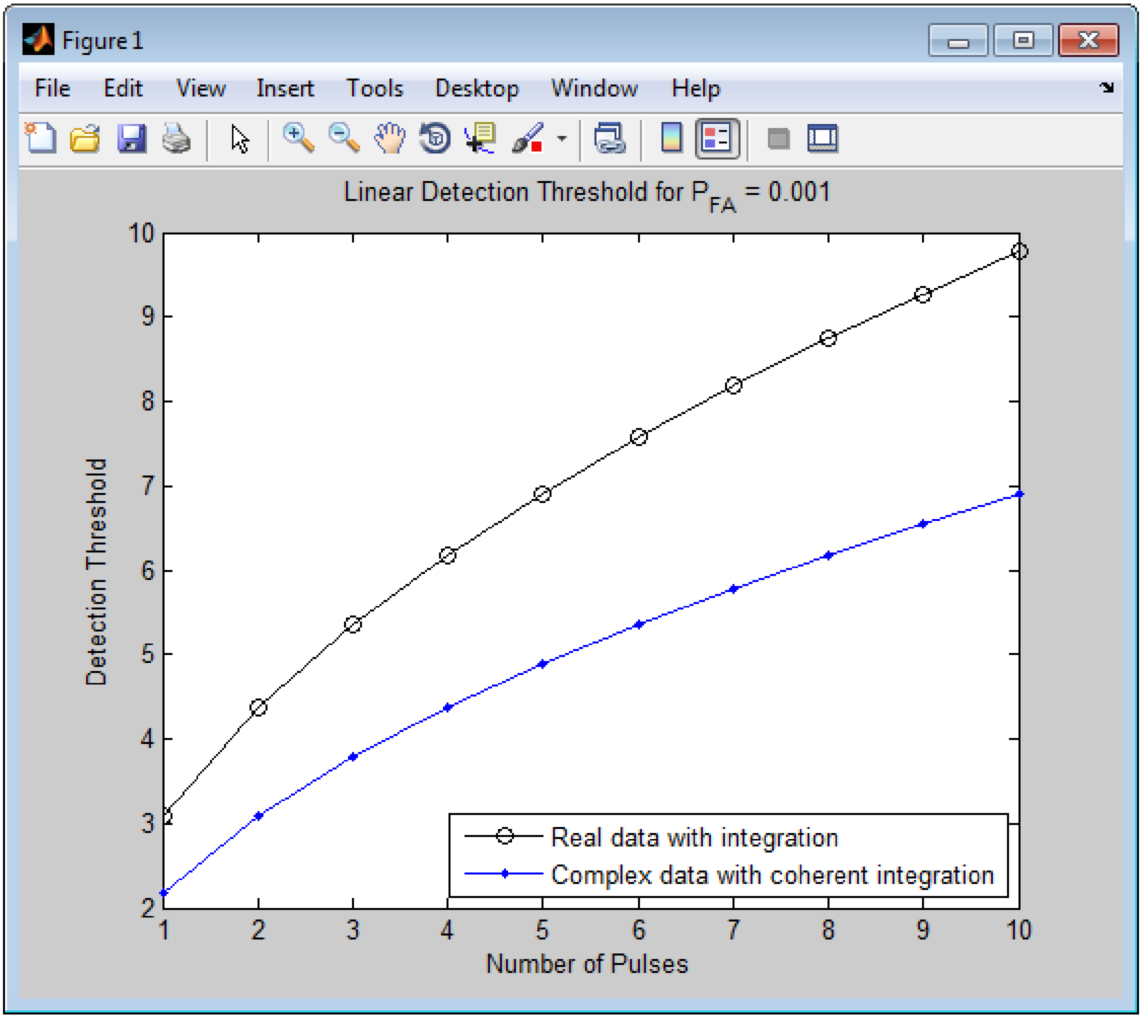
Plot the SNR threshold against the number of pulses, for real and complex data. In each case, the SNR threshold achieves a probability of false alarm of 0.001.

```
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
```



Plot the linear detection threshold against the number of pulses, for real and complex data. In each case, the 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','linear');
    snrcoh(num)  = npwgnthresh(Pfa,num,'coherent','linear');
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('Detection Threshold');
str = sprintf('Linear Detection Threshold for P_F_A = %4.3f',Pfa);
title(str)
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`

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

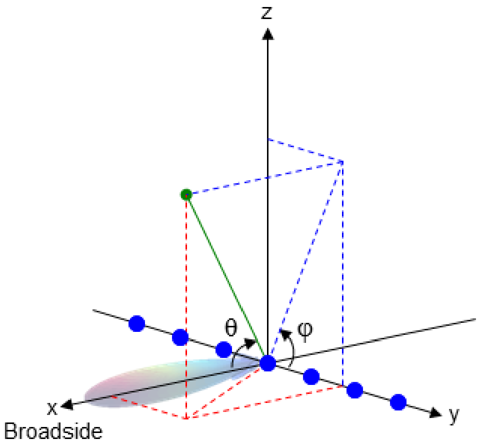
Azimuth and elevation angles, returned as a two-row matrix. Each column of the matrix represents an angle in degrees, in the form [azimuth; elevation]. The matrix dimensions of `AzEl` are the same as those of `PhiTheta`.

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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi \sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi \tan\theta$$

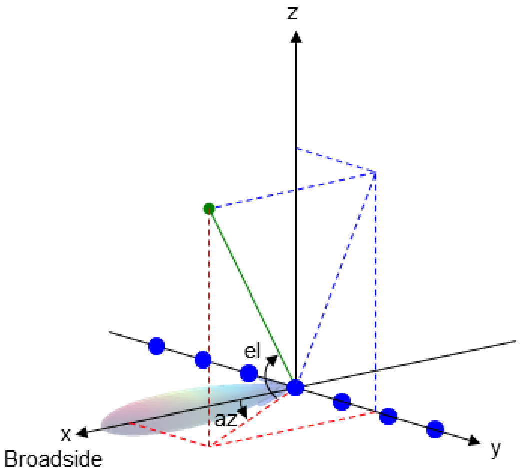
$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el}) / \sin(\mathrm{az})$$

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"

phitheta2azelpat

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

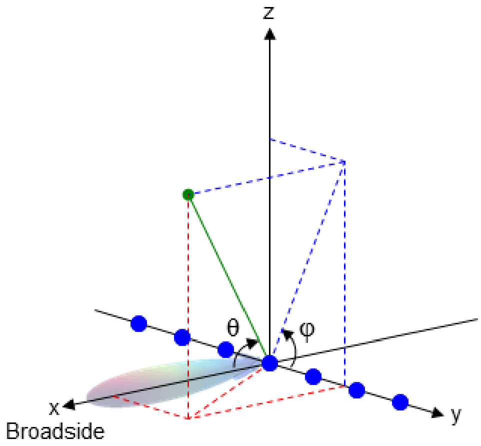
Elevation angles at which `pat_azel` samples the pattern, returned as a vector of length M. Angles are expressed in degrees.

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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi \sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi \tan\theta$$

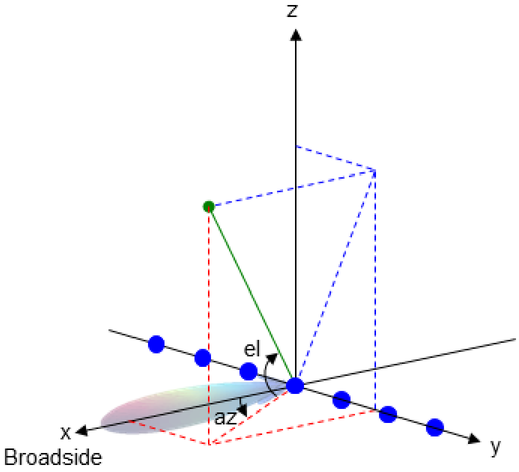
$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el}) / \sin(\mathrm{az})$$

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 Converted Radiation Pattern

Convert a radiation pattern from theta/phi coordinates to azimuth/elevation coordinates, with azimuth and elevation angles spaced 1° apart.

Define the pattern in terms of phi, ϕ, and theta, θ, coordinates.

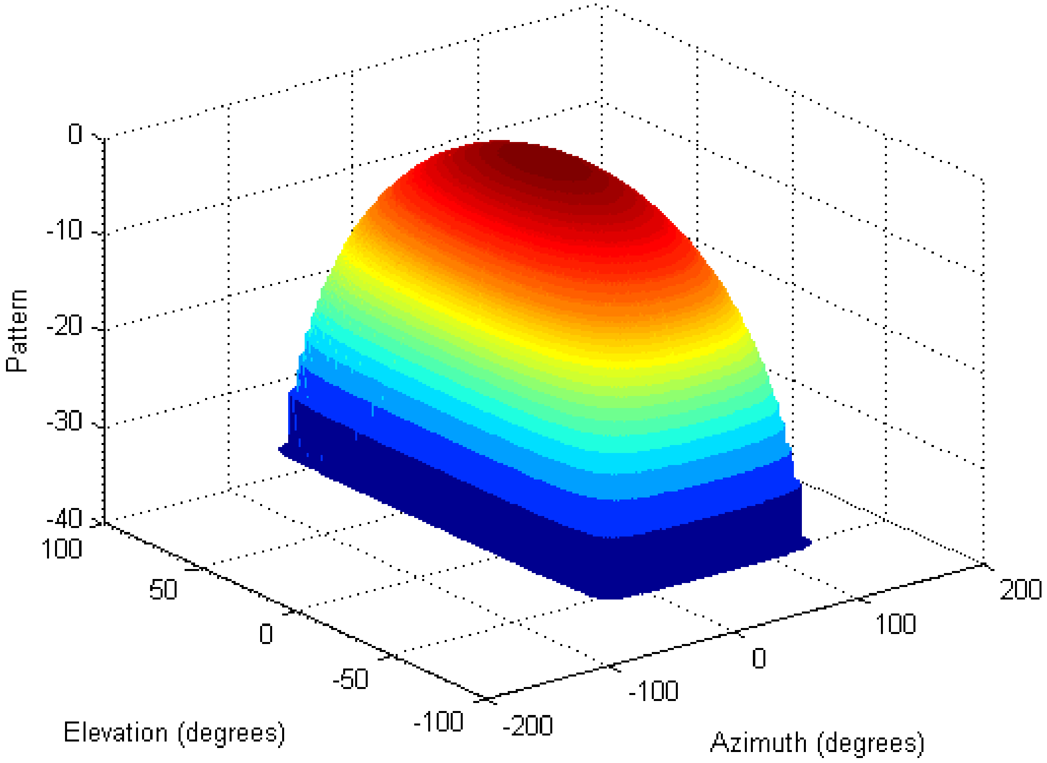
```
phi = 0:360;
theta = 0:180;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Convert the pattern to azimuth/elevation coordinates. Get the azimuth and elevation angles for use in plotting.

```
[pat_azel,az,el] = phitheta2azelpat(pat_phitheta,phi,theta);
```

Plot the radiation pattern.

```
H = surf(az,el,pat_azel);
set(H,'LineStyle','none')
xlabel('Azimuth (degrees)');
ylabel('Elevation (degrees)');
zlabel('Pattern');
```



Convert Radiation Pattern For Specific Azimuth/Elevation Values

Convert a radiation pattern from phi/theta coordinates to azimuth/elevation coordinates, with the azimuth and elevation angles spaced 5° apart.

Define the pattern in terms of phi and theta.

```
phi = 0:360;
```

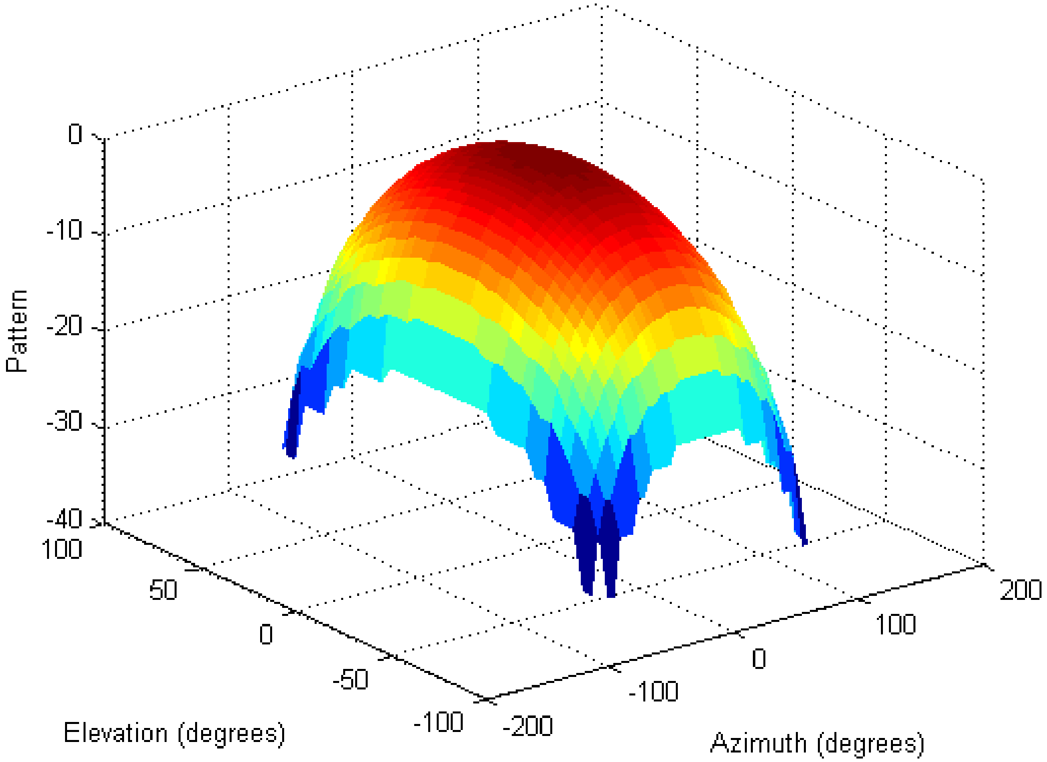
```
theta = 0:180;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Define the set of azimuth and elevation angles at which to sample the pattern. Then, convert the pattern.

```
az = -180:5:180;
el = -90:5:90;
pat_azel = phitheta2azelpat(pat_phitheta,phi,theta,az,el);
```

Plot the radiation pattern.

```
H = surf(az,el,pat_azel);
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”

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

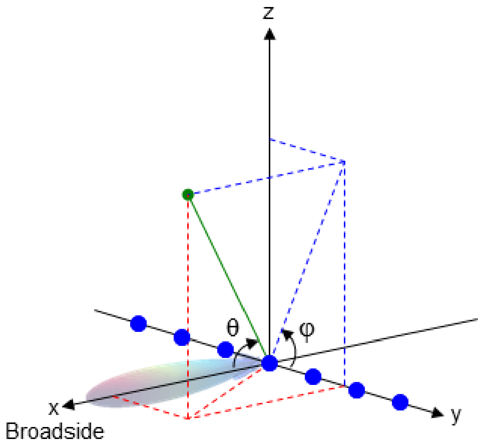
Angle in *u*/*v* space, returned as a two-row matrix. Each column of the matrix represents an angle in the form [*u*; *v*]. The matrix dimensions of `UV` are the same as those of `PhiTheta`.

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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\text{el}) = \sin\phi \sin\theta$$
$$\tan(\text{az}) = \cos\phi \tan\theta$$

$$\cos\theta = \cos(\text{el})\cos(\text{az})$$
$$\tan\phi = \tan(\text{el}) / \sin(\text{az})$$

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.

In terms of azimuth and elevation, the u and v coordinates are

$u = \cos(\mathrm{el}) \sin(\mathrm{az})$

$v = \sin(\mathrm{el})$

The values of u and v satisfy the inequalities

$-1 \leq u \leq 1$

$-1 \leq v \leq 1$

$u^2 + v^2 \leq 1$

Conversely, the phi and theta angles can be written in terms of u and v

$\tan(\varphi) = v/u$

$\sin(\theta) = \mathrm{sqrt}(\mathrm{u}^2 + \mathrm{v}^2)$

The azimuth and elevation angles can also be written in terms of u and v

$\sin(\mathrm{el}) = v$

$\tan(\mathrm{az}) = \mathrm{u}/\mathrm{sqrt}(1 - \mathrm{u}^2 - \mathrm{v}^2)$

Examples

Conversion of Phi/Theta Pair

Find the corresponding *u*/*v* representation for φ = 30 degrees and θ = 0 degrees.

```
UV = phitheta2uv([30; 0]);
```

See Also

`uv2phitheta`

Concepts

- “Spherical Coordinates”

phitheta2uvpat

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

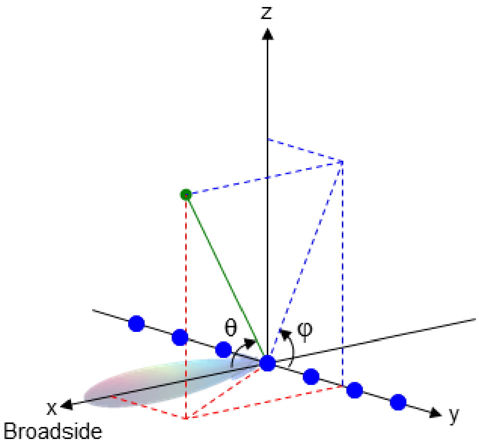
v coordinates at which `pat_uv` samples the pattern, returned as a vector of length M.

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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi \sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi \tan\theta$$

$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el}) / \sin(\mathrm{az})$$

U/V Space

The u and v coordinates are the direction cosines of a vector with respect to the y-axis and z-axis, respectively.

The u/v coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, 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.

In terms of azimuth and elevation, the u and v coordinates are

$$u = \cos(\mathrm{el}) \sin(\mathrm{az})$$

$$v = \sin(\mathrm{el})$$

The values of u and v satisfy the inequalities

$$-1 \le u \le 1$$

$$-1 \le v \le 1$$

$$u^2 + v^2 \le 1$$

Conversely, the phi and theta angles can be written in terms of u and v

$$\tan(\varphi) = v/u$$

$$\sin(\theta) = \mathrm{sqrt}(u^2 + v^2)$$

The azimuth and elevation angles can also be written in terms of u and v

$$\sin(\mathrm{el}) = v$$

$$\tan(\mathrm{az}) = u/\mathrm{sqrt}(1 - u^2 - v^2)$$

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);
```

Convert and Plot Radiation Pattern

Convert a radiation pattern to $u - v$ coordinates, with the u and v coordinates spaced by 0.01.

Define the pattern in terms of ϕ and θ.

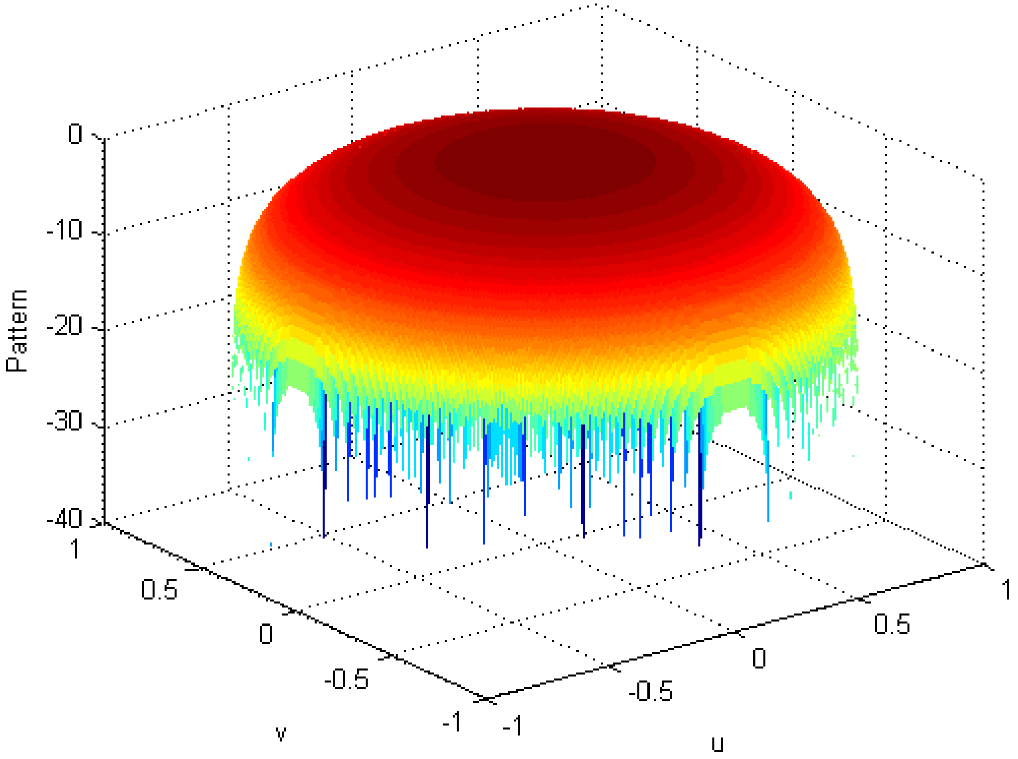
```
phi = 0:360;
theta = 0:90;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Convert the pattern to $u - v$ coordinates. Store the u and v coordinates for use in plotting.

```
[pat_uv,u,v] = phitheta2uvpat(pat_phitheta,phi,theta);
```

Plot the result.

```
H = surf(u,v,pat_uv);
set(H,'LineStyle','none')
xlabel('u');
ylabel('v');
zlabel('Pattern');
```



Convert Radiation Pattern For Specific U/V Values

Convert a radiation pattern to $u - v$ coordinates, with the u and v coordinates spaced by 0.05.

Define the pattern in terms of ϕ and θ .

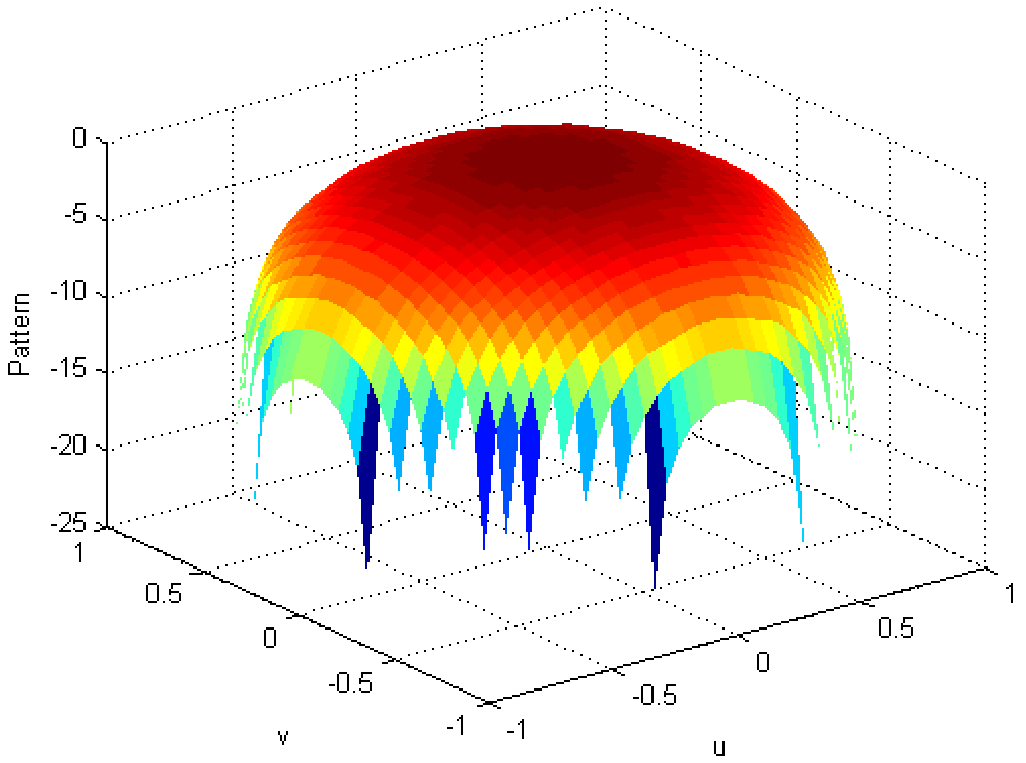
```
phi = 0:360;
theta = 0:90;
pat_phitheta = mag2db(repmat(cosd(theta)',1,numel(phi)));
```

Define the set of u and v coordinates at which to sample the pattern. Then, convert the pattern.

```
u = -1:0.05:1;
v = -1:0.05:1;
pat_uv = phitheta2uvpat(pat_phitheta,phi,theta,u,v);
```

Plot the result.

```
H = surf(u,v,pat_uv);
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 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

Convert linear component representation of field to circular component representation

Syntax

`cfv = pol2circpol(fv)`

Description

`cfv = pol2circpol(fv)` converts the linear polarization components of the field or fields contained in `fv` to their equivalent circular polarization components in `cfv`. The expression of a field in terms of a two-row vector of linear polarization components is called the *Jones vector formalism*.

Input Arguments

fv - Field vector in linear component representation

1-by-*N* complex-valued row vector or a 2-by-*N* complex-valued matrix

Field vector in its linear component representation specified as a 1-by-*N* complex row vector or a 2-by-*N* complex matrix. If `fv` is a matrix, each column in `fv` represents a field in the form of `[Eh;Ev]`, where `Eh` and `Ev` are the field's horizontal and vertical polarization components. If `fv` is a vector, each entry in `fv` is assumed to contain the polarization ratio, `Ev/Eh`. For a row vector, the value `Inf` designates the case when the ratio is computed for a field with `Eh = 0`.

Example: `[1;-i]`

Example: `2 + pi/3*i`

Data Types
`double`
Complex Number Support: Yes

Output Arguments

cfv - Field vector in circular component representation

1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix

Field vector in circular component representation returned as a 1-by-*N* complex-valued row vector or 2-by-*N*complex-valued matrix. `cfv` has the same dimensions as `fv`. If `fv` is a matrix, each column of `cfv` contains the circular polarization components, `[El;Er]`, of the field where `El` and `Er` are the left-circular and right-circular polarization components. If `fv` is a row vector, then `cfv` is also a row vector and each entry in `cfv` contains the circular polarization ratio, defined as `Er/El`.

Examples

Circular Polarization Components from Linear Polarization Components

Express a 45° linear polarized field in terms of right-circular and left-circular components.

```
fv = [2;2]
cfv = pol2circpol(fv)

cfv =

   1.4142 - 1.4142i
   1.4142 + 1.4142i
```

Circular Polarization Components from Linear Polarization Components for Two Fields

Specify two input fields `[1+1i;-1+1i]` and `[1;1]` in the same matrix. The first field is a linear representation of a left-circularly polarized field and the second is a linearly polarized field.

```
 fv=[1+1i 1;-1+1i 1]
 cfv = pol2circpol(fv)

cfv =

   1.4142 + 1.4142i   0.7071 - 0.7071i
   0.0000 + 0.0000i   0.7071 + 0.7071i
```

References

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

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

See Also

`circpol2pol` | `polellip` | `polratio` | `stokes`

polellip

Purpose

Parameters of ellipse traced out by tip of a polarized field vector

Syntax

```
tau = polellip(fv)
[tau,epsilon] = polellip(fv)
[tau,epsilon,ar] = polellip(fv)
[tau,epsilon,ar,rs] = polellip(fv)

polellip(fv)
```

Description

`tau = polellip(fv)` returns the tilt angle, in degrees, of the polarization ellipse of a field or set of fields specified in `fv`. `fv` contains the linear polarization components of a field in either one of two forms: (1) each column represents a field in the form of `[Eh;Ev]`, where `Eh` and `Ev` are the field's horizontal and vertical linear polarization components or (2) each column contains the polarization ratio, `Ev/Eh`. The expression of a field in terms of a two-row vector of linear polarization components is called the *Jones vector formalism*.

`[tau,epsilon] = polellip(fv)` returns, in addition, a row vector, `epsilon`, containing the ellipticity angle (in degrees) of the polarization ellipses. The ellipticity angle is the angle determined by the ratio of the length of the semi-minor axis to semi-major axis and lies in the range `[-45 ,45 ]`. This syntax can use any of the input arguments in the previous syntax.

`[tau,epsilon,ar] = polellip(fv)` returns, in addition, a row vector, `ar`, containing the axial ratios of the polarization ellipses. The axial ratio is defined as the ratio of the lengths of the semi-major axis of the ellipse to the semi-minor axis. This syntax can use any of the input arguments in the previous syntaxes.

`[tau,epsilon,ar,rs] = polellip(fv)` returns, in addition, a cell array of strings `rs`, containing the rotation senses of the polarization ellipses. Each entry in the array is one of `'Linear'`, `'Left Circular'`, `'Right Circular'`, `'Left Elliptical'` or `'Right Elliptical'`. This syntax can use any of the input arguments in the previous syntaxes.

`polellip(fv)` plots the polarization ellipse of the field specified in `fv`. This syntax requires that `fv` have only one column. Unlike the returned arguments, the size of the drawn ellipse depends upon the magnitude of `fv`.

Input Argument

fv - Field vector in linear component representation

1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix

Field vector in linear component representation specified as a 1-by-*N* complex-valued row vector or 2-by-*N* complex-valued matrix. Each column contains an instance of a field specification. If `fv` is a matrix, each column in `fv` represents a field in the form of `[Eh;Ev]`, where `Eh` and `Ev` are the field's linear horizontal and vertical polarization components. If `fv` is a row vector, then the row contains the ratio of the vertical to horizontal components of the field `Ev/Eh`. For a row vector, the value `Inf` is allowed to designate the case when the ratio is computed for `Eh = 0`. `Eh` and `Ev` cannot both be set to zero.

Example: `[1;-i]`

Example: `2 + pi/3*i`

Data Types
`double`
Complex Number Support: Yes

Output Arguments

tau - Tilt angle of polarization ellipse

1-by-*N* real-valued row vector

Tilt angle of polarization ellipse returned as a 1-by-*N* real-valued row vector. Each entry in `tau` contains the tilt angle of the polarization ellipse associated with each column of the field `fv`. The tilt angle is the angle between the semi-major axis of the ellipse and the horizontal axis (i.e. *x*axis) and lies in the range `[-90,90]` .

epsilon - Ellipticity angle of the polarization ellipse

1-by-*N* real-valued row vector

Ellipticity angle of the polarization ellipse returned as 1-by-*N* real-valued row vector. Each entry in `epsilon` contains the ellipticity angle of the polarization ellipse associated with each column of the field `fv`. The ellipticity angle describes the shape of the ellipse and lies in the range `[-45 ,45 ]`.

ar - Axial ratio of the polarization ellipse

1-by-*N* real-valued row vector

Axial ratio of the polarization ellipse returned as a 1-by-*N* real-valued row vector. Each entry in `ar` contains the axial ratio of the polarization ellipse associated with each column of the field `fv`. The axial ratio is the signed ratio of the major-axis length to the minor-axis length of the polarization ellipse. Its absolute value is always greater than or equal to one. The sign of `ar` carries the rotational sense of the vector – a negative sign denotes left-handed rotation and a positive sign denotes right-handed rotation.

rs - Rotation sense of the polarization ellipse

1-by-*N* cell array of strings

Rotation sense of the polarization ellipse returned as a 1-by-*N* cell array of strings. Each entry in `rs` contains the rotation sense of the polarization ellipse associated with each column of the field `fv`. The rotation sense can be one of `'Linear'`, `'Left Circular'`, `'Right Circular'`, `'Left Elliptical'` or `'Right Elliptical'`.

Examples

Tilt Angle for Linearly Polarized Field

Create an input field that is linearly polarized by setting both the horizontal and vertical components to have the same phase.

```
fv = [2;1];
tau = polellip(fv)
```

```
tau =
   26.5651
```

For linear polarization, `tau`, can be computed from `tau=atan(fv(2)/fv(1))*180/pi` .

Tilt Angle and Ellipticity for Elliptically Polarized Field

Start with an elliptically polarized input field (the horizontal and vertical components differ in magnitude and in phase). Choose the phase difference to be 90°.

```
fv = [3*exp(-i*pi/2);1];
[tau,epsilon] = polellip(fv)

tau =

   2.3389e-15

epsilon =

   18.435
```

The tilt vanishes because of the 90° phase difference between the horizontal and vertical components of the field.

Tilt Angle, Ellipticity and Axial Ratio for Elliptically Polarized Field

Start with an elliptically polarized input field (the horizontal and vertical components differ in magnitude and in phase). Choose the phase difference to be 60°.

```
fv = [2*exp(-i*pi/3);1];
[tau,epsilon,ar] = polellip(fv)

tau =

   16.8450

epsilon =
   21.9269

ar =

   -2.4842
```

The nonzero tilt occurs because of the 60° phase difference. The negative value of `ar` signifies left elliptical polarization.

Tilt Angle, Ellipticity, Axial Ratio and Rotation Sense for Elliptically Polarized Field

Start with an elliptically polarized input field (the horizontal and vertical components differ in magnitude and in phase). Choose the phase difference to be 60°.

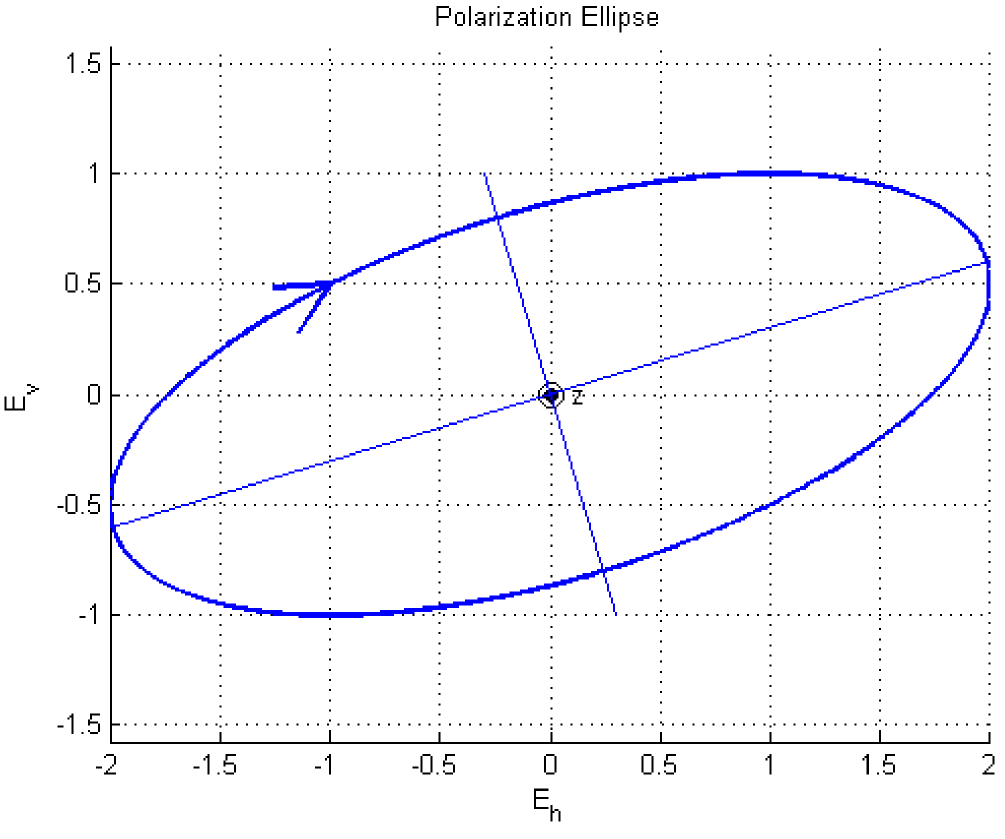
```
fv = [2*exp(-i*pi/3);1];
[tau,epsilon,ar,rs] = polellip(fv)

tau =

   16.8450

epsilon =

   21.9269

ar =

   -2.4842

rs =

    'Left Elliptical'
```

The nonzero tilt occurs because of the 60° phase difference and the rotation sense is `'Left Elliptical'` indicating that the tip of the field vector is moving clockwise when looking towards the source of the field.

Polarization Ellipse

Draw the figure of an elliptically polarized field. Begin with an elliptically polarized input field (the horizontal and vertical components differ in magnitude and in phase) and choose the phase difference to be 60 degrees.

```
fv = [2*exp(-i*pi/3);1];
polellip(fv)
```



The rotation sense is `'Left Elliptical'` as shown by the direction of the arrow on the ellipse. The filled circle at the origin indicates that the observer is looking towards the source of the field.

References

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

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

See Also

`circpol2pol` | `pol2circpol` | `polratio` | `stokes`

Purpose

Polarization loss

Syntax

```
rho = polloss(fv_tr,fv_rcv)
rho = polloss(fv_tr,fv_rcv,pos_rcv)
rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv)
rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv,pos_tr)
rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv,pos_tr,axes_tr)
```

Description

`rho = polloss(fv_tr,fv_rcv)` returns the loss, in decibels, because of mismatch between the polarization of a transmitted field, `fv_tr`, and the polarization of the receiving antenna, `fv_rcv`. The field vector lies in a plane orthogonal to the direction of propagation from the transmitter to the receiver. The transmitted field is represented as a 2-by-1 column vector `[Eh;Ev]`. In this vector, `Eh` and `Ev` are the field's horizontal and vertical linear polarization components with respect to the transmitter's local coordinate system. The receiving antenna's polarization is specified by a 2-by-1 column vector, `fv_rcv`. You can also specify this polarization in the form of `[Eh;Ev]` with respect to the receiving antenna's local coordinate system. In this syntax, both local coordinate axes align with the global coordinate system.

`rho = polloss(fv_tr,fv_rcv,pos_rcv)` specifies, in addition, the position of the receiver. The receiver is defined as a 3-by-1 column vector, `[x;y;z]`, with respect to the global coordinate system (position units are in meters). This syntax can use any of the input arguments in the previous syntax.

`rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv)` specifies, in addition, the orthonormal axes, `axes_rcv`. These axes define the receiver's local coordinate system as a 3-by-3 matrix. The first column gives the x-axis of the local system with respect to the global coordinate system. The second and third columns give the y and z axes, respectively. This syntax can use any of the input arguments in the previous syntaxes.

`rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv,pos_tr)` specifies, in addition, the position of the transmitter as a 3-by-1 column vector, `[x;y;z]`, with respect to the global coordinate system (position units are in meters). This syntax can use any of the input arguments in the previous syntaxes.

`rho = polloss(fv_tr,fv_rcv,pos_rcv,axes_rcv,pos_tr,axes_tr)` specifies, in addition, the orthonormal axes, `axes_tr`. These axes define the transmitter's local coordinate system as a 3-by-3 matrix. The first column gives the *x*-axis of the local system with respect to the global coordinate system. The second and third columns give the *y* and *z* axes, respectively. This syntax can use any of the input arguments in the previous syntaxes.

Input Arguments

fv_tr - Transmitted field vector in linear component representation

2-by-1 complex-valued column vector

The transmitted field vector in linear component representation specified as a 2-by-1, complex-valued column vector `[Eh;Ev]`. In this vector, `Eh` and `Ev` are the field's horizontal and vertical linear components.

Example: [1;1]

Data Types
`double`
Complex Number Support: Yes

fv_rcv - Receiver polarization vector in linear component representation

2-by-1 complex-valued column vector

Receiver polarization vector in linear component representation specified as a 2-by-1, complex-valued column vector `[Eh;Ev]`. In this vector, `Eh` and `Ev` are the polarization vector's horizontal and vertical linear components.

Example: [0;1]

Data Types
`double`
Complex Number Support: Yes

pos_rcv - Receiving antenna position

[0;0;0] (default) | 3-by-1 real-valued column vector

Receiving antenna position specified as a 3-by-1, real-valued column vector. The components of `pos_rcv` are specified in the global coordinate system as `[x;y;z]`.

Example: [1000;0;0]

Data Types
`double`

axes_rcv - Receiving antenna local coordinate axes

3-by-3 identity matrix (default) | 3-by-3 real-valued matrix

Receiving antenna local coordinate axes specified as a 3-by-3, real-valued matrix. Each column is a unit vector specifying the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively, with respect to the global coordinate system. Each column is written in `[x;y;z]` form. If `axes_rcv` is specified as the identity matrix, the local coordinate system is aligned with the global coordinate system.

Example: [1, 0, 0; 0, 1, 0; 0, 0 ,1]

Data Types
`double`

pos_tr - Transmitter position

[0;0;0] (default) | 3-by-1 real-valued column vector

Transmitter position specified as a 3-by-1, real-valued column vector. The components of `pos_tr` are specified in the global coordinate system as `[x;y;z]`.

Example: [0;0;0]

Data Types
`double`

axes_tr - Transmitting antenna local coordinate axes

3-by-3 identity matrix (default) | 3-by-3 real-valued matrix

Transmitting antenna local coordinate axes specified as a 3-by-3, real-valued matrix. Each column is a unit vector specifying the local coordinate system's orthonormal *x*, *y*, and *z* axes, respectively, with respect to the global coordinate system. Each column is written in `[x;y;z]` form. If `axes_tr` is the identity matrix, the local coordinate system is aligned with the global coordinate system.

Example: [1, 0, 0; 0, 1, 0; 0, 0 ,1]

Data Types
`double`

Output Arguments

rho - Polarization loss

scalar

Polarization loss returned as scalar in decibel units. The polarization loss is the projection of the normalized transmitted field vector into the normalized receiving antenna polarization vector. Its value lies between zero and unity. When converted into dB, (and a sign changed to show loss as positive) its value lies between 0 and `-Inf`.

Examples

Mismatch Between a 45° Polarized Field and a Horizontally Polarized Receiver

Begin with a 45° polarized transmitted field and a receiver that is horizontally polarized. By default, the transmitter and receiver local axes coincide with the global coordinate system. Compute the polarization loss in dB.

```
fv_tr = [1;1];
fv_rcv = [1;0];
rho = polloss(fv_tr,fv_rcv);
rho =

    3.0103
```

The loss is 3 dB as expected because only half the power of the field matches to the receive antenna polarization.

No Polarization Loss Because of Receiver Motion

Begin with identical transmitter and receiver polarizations. Place the receiver at a position 100 meters along the *y*-axis. The transmitter is at the origin (its default position) and both local coordinate axes coincide with the global coordinate system (by default). First, compute the polarization loss. Then, move the receiver 100 meters along the *x*-axis, and compute the polarization loss again.

```
fv_tr = [1;0];
fv_rcv = [1;0];
pos_rcv = [0;100;0];
rho(1) = polloss(fv_tr,fv_rcv,pos_rcv);
pos_rcv = [100;100;0];
rho(2) = polloss(fv_tr,fv_rcv,pos_rcv);
```

```
rho =

     0     0
```

No polarization loss occurs at either position. The spherical basis vectors of each antenna are parallel to their counterparts and the polarization vectors are the same.

Loss Because of Receiver Axes Rotation

Start with identical transmitter and receiver polarizations. Put the receiver at a position 100 meters along the *y*-axis. The transmitter is at the origin (default) and both local coordinate axes coincide with the global coordinate system (default). Compute the loss, and then rotate the receiver 30° around the *y*-axis. This rotation changes the azimuth and elevation of the transmitter with respect to the receiver and, therefore, the direction of polarization.

```
fv_tr = [1;0];
fv_rcv = [1;0];
pos_rcv = [0;100;0];
ax_rcv = azelaxes(0,0);
rho(1) = polloss(fv_tr,fv_rcv,pos_rcv,ax_rcv);
ax_rcv = roty(30)*ax_rcv;
rho(2) = polloss(fv_tr,fv_rcv,pos_rcv,ax_rcv);
```

```
rho =

         0    1.2494
```

The receiver polarization vector remains unchanged. However, rotating the local coordinate system changes the direction of the field of the receiving antenna polarization with respect to global coordinates. This change results in a 1.2 dB loss.

No Polarization Loss Because of Transmitter Motion

Start with identical transmitter and receiver polarizations. Put the receiver at a position 100 meters along the *y*-axis. The transmitter is at the origin (default) and both local coordinate axes coincide with the global coordinate system (default). First, compute the polarization loss. Then, move the transmitter 100 meters along the *x*-axis and 100 meters along the *y*-axis, and compute the polarization loss again.

```
fv_tr = [1;0];
fv_rcv = [1;0];
pos_rcv = [0;100;0];
ax_rcv = azelaxes(0,0);
pos_tr = [0;0;0];
rho(1) = polloss(fv_tr,fv_rcv,pos_rcv,ax_rcv,pos_tr);
pos_tr = [100;100;0];
rho(2) = polloss(fv_tr,fv_rcv,pos_rcv,ax_rcv,pos_tr);
```

```
rho =

     0     0
```

There is no polarization loss at either position because the spherical basis vectors of each antenna are parallel to their counterparts and the polarization vectors are the same.

Plot Polarization Loss as Receiving Antenna Rotates

Specifying identical transmitter and receiver polarizations, plot the loss as the local receiving antenna axes rotate around the x -axis.

```
fv_tr = [1;0];
fv_rcv = [1;0];
```

The position of the transmitting antenna is at the origin and its local axes align with the global coordinate system. The position of the receiving antenna is 100 meters along the global x -axis. However, its local x -axis points towards the transmitting antenna.

```
pos_tr = [0;0;0];
axes_tr = azelaxes(0,0);
pos_rcv = [100;0;0];
axes_rcv0 = rotz(180)*azelaxes(0,0);
```

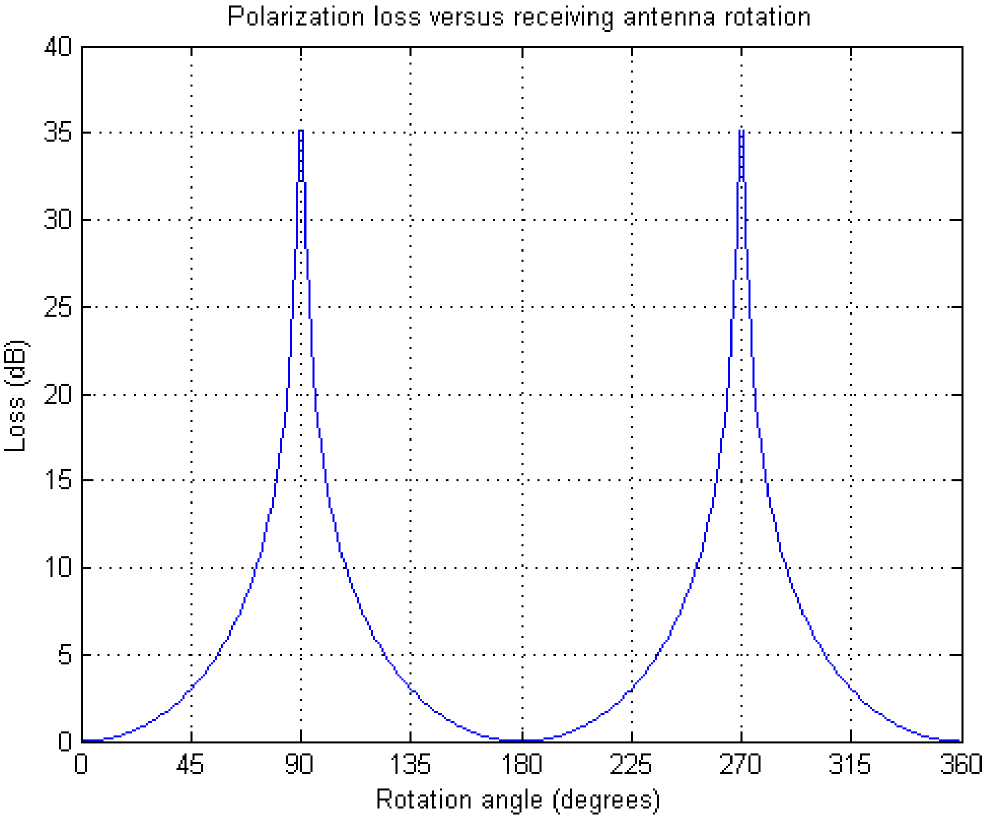
Rotate the receiving antenna around its local x -axis in one-degree increments. Compute the loss for each angle.

```
angles = [0:1:359];
n = size(angles,2);
rho = zeros(1,n); % Initialize space
for k = 1:n
    axes_rcv = rotx(angles(k))*axes_rcv0;
    rho(k) = polloss(fv_tr,fv_rcv,pos_tr,axes_tr,...
        pos_rcv,axes_rcv);
end
```

Plot the polarization loss.

```
hp = plot(angles,rho);  hax = get(hp,'parent');
set(hax,'xlim',[0,360]);
xticks = (0:(n-1))*45;
```

```
set(hax,'xtick',xticks);
grid;
title('Polarization loss versus receiving antenna rotation')
xlabel('Rotation angle (degrees)'); ylabel('Loss (dB)');
```



The angle-loss plot shows nulls (`Inf` dB) at 90 degrees and 270 degrees where the polarizations are orthogonal.

Definitions

Polarization Loss Due to Field and Receiver Mismatch

Loss occurs when a receiver is not matched to the polarization of an incident electromagnetic field.

In the case of the polarization of a field emitted by a transmitting antenna, first, look at the far zone of the transmitting antenna, as shown in the following figure. At this location—which is the location of the receiving antenna—the electromagnetic field is orthogonal to the direction from transmitter to receiver.

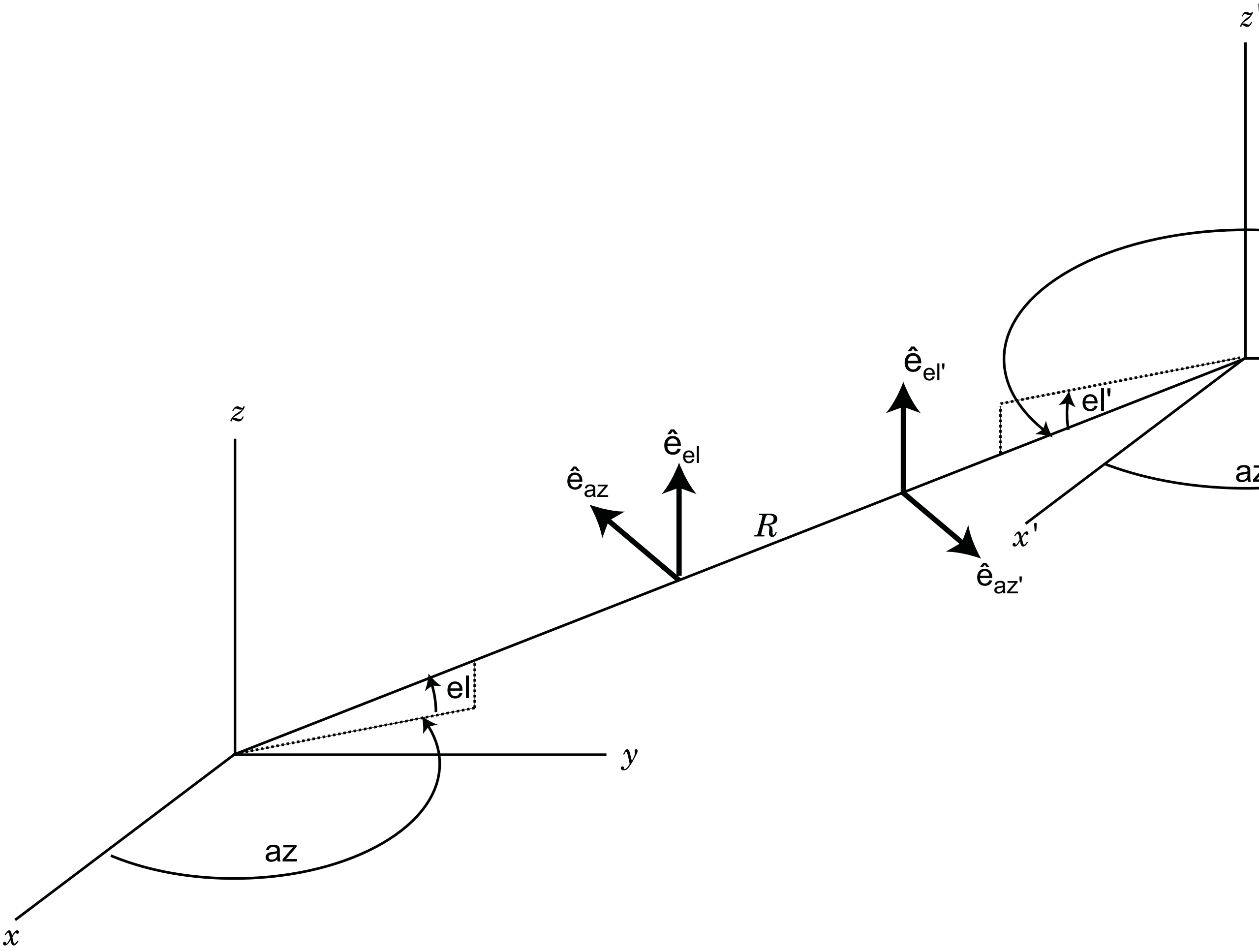
You can represent the transmitted electromagnetic field, `fv_tr`, by the components of a vector with respect to a spherical basis of the transmitter's local coordinate system. The orientation of this basis depends on its direction from the origin. The direction is specified by the azimuth and elevation of the receiving antenna with respect to the transmitter's local coordinate system. Then, the transmitter's polarization, in terms of the spherical basis vectors of the transmitter's local coordinate system, is

$$\mathbf{E} = E_H \hat{\mathbf{e}}_{az} + E_V \hat{\mathbf{e}}_{el} = E_m \mathbf{P}_i$$

In the same manner, the receiver's polarization vector, `fv_rcv`, is defined with respect to a spherical basis in the receiver's local coordinate system. Now, the azimuth and elevation specify the transmitter's position with respect to the receiver's local coordinate system. You can write the receiving antennas polarization in terms of the spherical basis vectors of the receiver's local coordinate system:

$$\mathbf{P} = P_H \hat{\mathbf{e}}'_{az} + P_V \hat{\mathbf{e}}'_{el}$$

This figure shows the construction of the different transmitter and receiver local coordinate systems. It also shows the spherical basis vectors with which to write the field components.



The polarization loss is the projection (or dot product) of the normalized transmitted field vector onto the normalized receiver polarization vector. Notice that the loss occurs because of the mismatch in direction of the two vectors not in their magnitudes. Because the vectors are defined in different coordinate systems, they must be converted to the global coordinate system in order to form the projection. The polarization loss is defined by:

$$\rho = \frac{|\mathbf{E}_i \cdot \mathbf{P}|^2}{|\mathbf{E}_i|^2 |\mathbf{P}|^2}$$

References

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

See Also

`polellip` | `stokes`

polratio

Purpose

Ratio of vertical to horizontal linear polarization components of a field

Syntax

`p = polratio(fv)`

Description

`p = polratio(fv)` returns the ratio of the vertical to horizontal component of the field or set of fields contained in `fv`.

Each column of `fv` contains the linear polarization components of a field in the form `[Eh;Ev]`, where `Eh` and `Ev` are the field's linear horizontal and vertical polarization components. The expression of a field in terms of a two-row vector of linear polarization components is called the *Jones vector formalism*. The argument `fv` can refer to either the electric or magnetic part of an electromagnetic wave.

Each entry in `p` contains the ratio `Ev/Eh` of the components of `fv`.

Input Arguments

fv - Field vector in linear component representation

2-by-N complex-valued matrix

Field vector in linear component representation specified as a 2-by-N complex-valued matrix. Each column of `fv` contains an instance of a field specified by `[Eh;Ev]`, where `Eh` and `Ev` are the field's linear horizontal and vertical polarization components. Two rows of the same column cannot both be zero.

Example: [2 , i; i, 1]

Data Types
`double`
Complex Number Support: Yes

Output Arguments

p - Polarization ratio

1-by-N complex-valued row vector

Polarization ratio returned as a 1-by-N complex-valued row vector. `p` contains the ratio of the components of the second row of `fv` to the first row, `Ev/Eh`.

Examples

Polarization Ratio for 45° Linearly Polarized Field

Determine the polarization ratio for a linearly polarized field (when the horizontal and vertical components of a field have the same phase).

```
fv = [2 ; 2];
p = polratio(fv)

p =

         1
```

The resulting polarization ratio is real. The components also have equal amplitudes so the polarization ratio is unity.

Polarization Ratios for Two Fields

Pass two fields via a single matrix. The first field is `[2;i]`, while the second is `[i;1]`.

```
fv = [2 , i; i, 1];
p = polratio(fv)

p =

        0 + 0.5000i        0 - 1.0000i
```

Polarization Ratio for Vertically Polarized Field

Determine the polarization ratio for a vertically polarized field (when the horizontal component of the field vanishes).

```
fv = [0 ; 2];
p = polratio(fv)

p =

   Inf
```

The polarization ratio is infinite as expected from `Ev/Eh`.

References

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

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

See Also

`circpol2pol` | `pol2circpol` | `polellip` | `stokes`

Purpose

Copolarization and cross-polarization signatures

Syntax

```
resp = polsignature(rcsmat)
resp = polsignature(rcsmat,type)
resp = polsignature(rcsmat,type,epsilon)
resp = polsignature(rcsmat,type,epsilon,tau)

polsignature(___)
```

Description

`resp = polsignature(rcsmat)` returns the normalized radar cross-section copolarization (*co-pol*) signature, `resp` (in square meters), determined from the scattering cross section matrix, `rcsmat` of an object. The signature is a function of the transmitting antenna polarization, specified by the ellipticity angle and the tilt angle of the polarization ellipse. In this syntax case, the ellipticity angle takes the values `[-45:45]` and the tilt angle takes the values `[-90:90].` The output `resp` is a 181-by-91 matrix whose elements correspond to the signature at each ellipticity angle-tilt angle pair.

`resp = polsignature(rcsmat,type)`, in addition, specifies the polarization signature type as one of `'c'|'x'`, where `'c'` creates the copolarization signature and `'x'` creates the cross-polarization (*cross-pol*) signature. The default value of this parameter is `'c'`. The output `resp` is a 181-by-91 matrix whose elements correspond to the signature at each ellipticity angle-tilt angle pair. This syntax can use the input arguments in the previous syntax.

`resp = polsignature(rcsmat,type,epsilon)`, in addition, specifies the transmit antenna polarization's ellipticity angle (in degrees) as a length-M vector. The angle `epsilon` must lie between –45° and 45°. The argument `resp` is a 181-by-M matrix whose elements correspond to the signature at each ellipticity angle-tilt angle pair. This syntax can use any of the input arguments in the previous syntaxes.

`resp = polsignature(rcsmat,type,epsilon,tau)`, in addition, specifies the tilt angle of the polarization ellipse of the transmitted wave (in degrees) as a length-*N* vector. The angle `tau` must be between –90° and 90°. The signature, `resp`, is represented as a function of the transmitting antenna polarization. The transmitting antenna polarization is characterized by the ellipticity angle, `epsilon`, and the tilt angle, `tau`. The argument `resp` is a *N*-by-*M* matrix whose elements correspond to the signature at each ellipticity angle-tilt angle pair. This syntax can use any of the input arguments in the previous syntaxes.

`polsignature( ___ )` plots a three dimensional surface using any of the syntax forms specified above.

Input Arguments

rcsmat - Radar cross-section scattering matrix

2-by-2 complex-valued matrix

Radar cross-section scattering matrix (*RCSM*) of an object specified as a 2-by-2, complex-valued matrix. The radar cross-section scattering matrix describes the polarization of a scattered wave as a function of the polarization of an incident wave upon a target. The response to an incident wave can be construct from the individual responses to the incident field's horizontal and vertical polarization components. These components are taken with respect to the transmit antenna or array local coordinate system. The scattered wave can be decomposed into horizontal and vertical polarization components with respect to the receive antenna or array local coordinate system. The matrix *RCSM* contains four components `[rcs_hh rcs_hv;rcs_vh rcs_vv]` where each component is the radar cross section defined by the polarization of the transmit and receive antennas.

- `rcs_hh` – Horizontal response due to horizontal transmit polarization component
- `rcs_hv` – Horizontal response due to vertical transmit polarization component
- `rcs_vh` – Vertical response due to horizontal transmit polarization component

- `rcs_vv` – Vertical response due to vertical transmit polarization component

In the monostatic radar case, when the wave is backscattered, the RCSM matrix is symmetric.

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

Data Types
`double`
Complex Number Support: Yes

type - Polarization signature type

`'c'` (default) | Single character `'c'`|`'x'`

Polarization signature type of the scattered wave specified by a single character: `'c'` denoting the copolarized signature or `'x'` denoting the cross-polarized signature.

Example: `'x'`

Data Types
`char`

epsilon - Ellipticity angle of the polarization ellipse of the transmitted wave

`[-45:45]` (default) | scalar or 1-by-M real-valued row vector

Ellipticity angle of the polarization ellipse of the transmitted wave specified as a length-M vector. Units are degrees. The ellipticity angle describes the shape of the ellipse. By definition, the tangent of the ellipticity angle is the signed ratio of the semiminor axis to semimajor axis of the polarization ellipse. Since the absolute value of this ratio cannot exceed unity, the ellipticity angle lies between ±45°.

Example: `[-45:0.5:45]`

Data Types
`double`

tau - Tilt angle of the polarization ellipse of the transmitted wave

`[-90:90]` (default) | scalar or 1-by-*N* real-valued row vector.

Tilt angle of the polarization ellipse of the transmitted wave specified as a length-*N* vector. Units are degrees. The tilt angle is defined as the angle between the semimajor axis of the ellipse and the *x*-axis. Because the ellipse is symmetrical, an ellipse with a tilt angle of 100° is the same ellipse as one with a tilt angle of –80°. Therefore, the tilt angle need only be specified between ±90°.

Example: `[-30:2:30]`

Data Types
`double`

Output Arguments

resp - Normalized magnitude response

scalar or *N*-by-*M* real-valued matrix.

Normalized magnitude response returned as a scalar or *N*-by-*M*, real-valued matrix having values between 0 and 1. `resp` returns a value for each ellipticity-tilt angle pair.

Examples

Copolarization Signature of a Dihedral

Use the default values of this function to create a matrix of copolarization responses to a dihedral object. Specify the ellipticity angle values as `[-45:45]` and the tilt angle values as `[-90:90]`. Then, draw the response matrix as an image.

```
rscmat = [-1,0;0,1];
resp = polsignature(rscmat);
el = [-45:45];
tilt = [-90:90];
imagesc(el,tilt,resp); ylabel('Tilt (degrees)');
xlabel('Ellipticity Angle (degrees)'), axis image
set(gca,'xtick',[-45:15:45],'ytick',[-90:15:90]);
title('Co-polarization signature of dihedral');
colorbar;
```

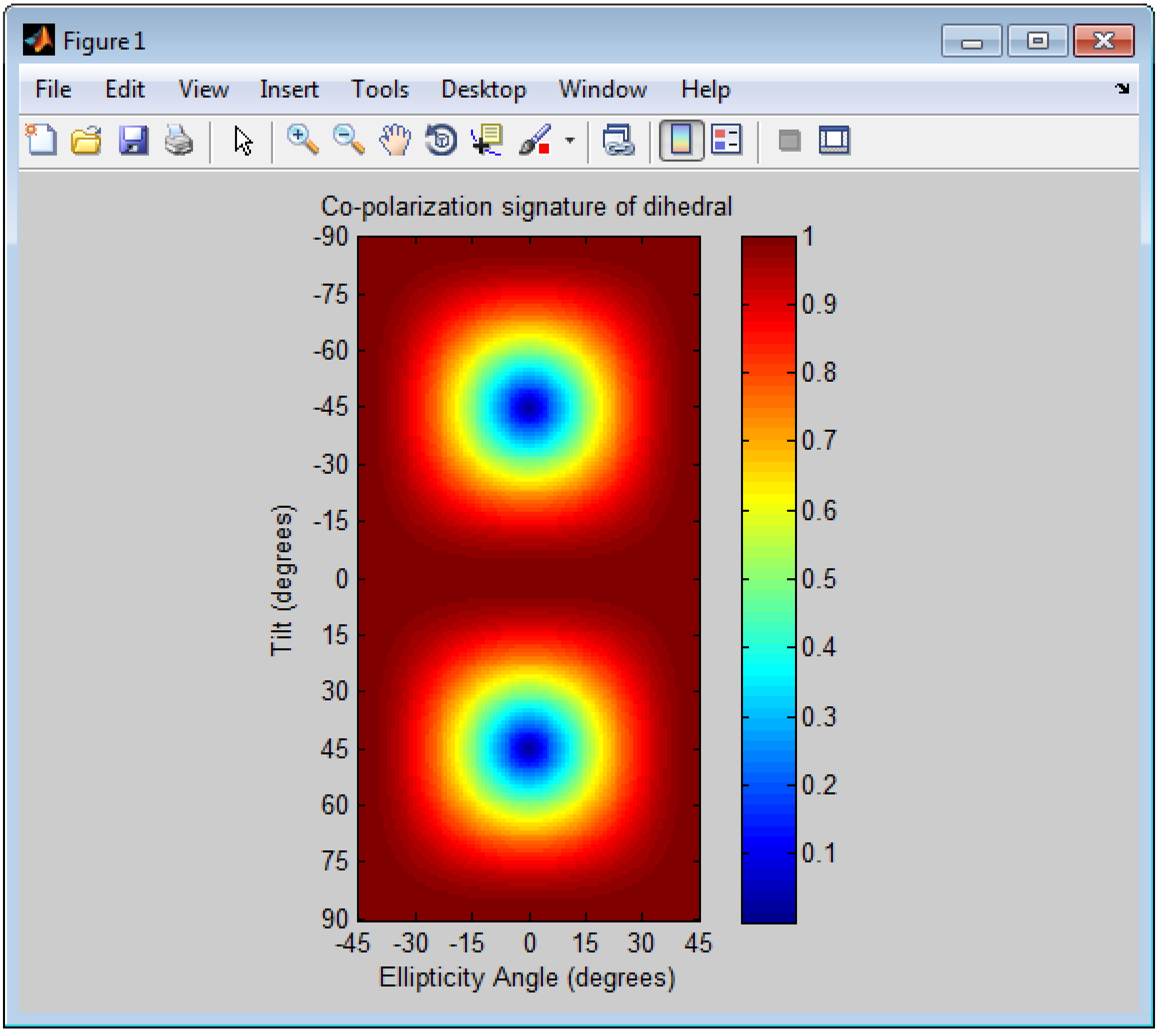


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Co-polarization signature of dihedral
Tilt (degrees)
-90
-75
-60
-45
-30
-15
0
15
30
45
60
75
90
Ellipticity Angle (degrees)
-45
-30
-15
0
15
30
45
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1

Cross-Polarization Signature of a Dihedral

Set the `type` argument to `'x'` to create a cross-polarization response matrix for a dihedral object. Use the default values of ellipticity angles,

`[-45:45]`, and tilt angles, `[-90:90]`. Then, draw the response matrix as an image.

```
rscmat = [-1,0;0,1];
resp = polsignature(rscmat,'x');
el = [-45:45];
tilt = [-90:90];
imagesc(el,tilt,resp); ylabel('Tilt (degrees)');
xlabel('Ellipticity Angle (degrees)'), axis image
set(gca,'xtick',[-45:15:45],'ytick',[-90:15:90]);
title('Cross-polarization signature of dihedral');
colorbar;
```

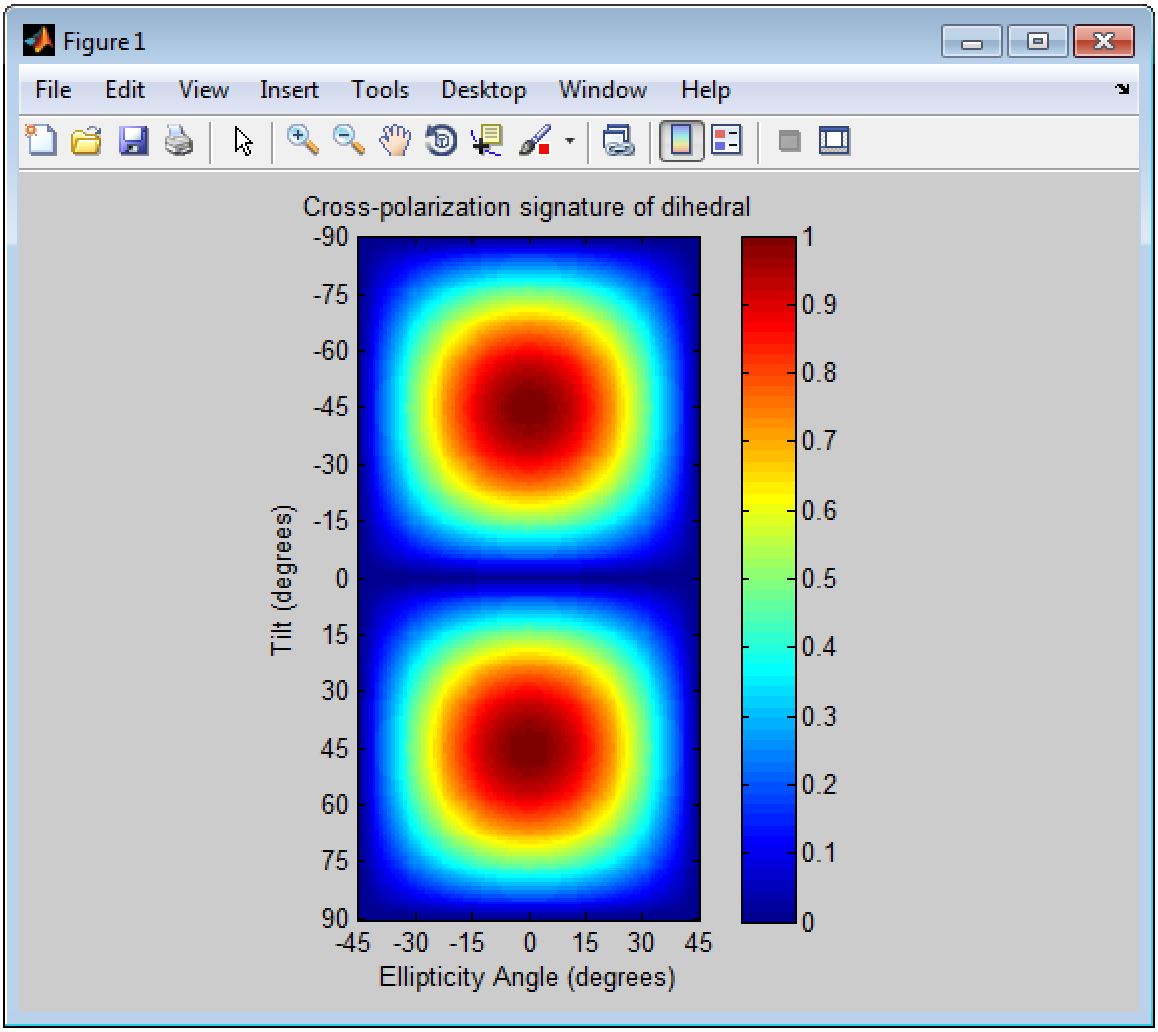


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Cross-polarization signature of dihedral
-90
-75
-60
-45
-30
-15
0
15
30
45
60
75
90
Tilt (degrees)
-45
-30
-15
0
15
30
45
Ellipticity Angle (degrees)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Signatures for Linear Polarization with Varied Tilt Angles

Set the ellipticity angle to zero, and vary the tilt angle from -90° to +90° to generate all possible linear polarization directions. Then, plot both the copolarization and cross-polarization signatures.

```
rscmat = [-1,0;0,1];
el = [0];
respc = polsignature(rscmat,'c',el);
respx = polsignature(rscmat,'x',el);
tilt = [-90:90];
plot(tilt,respc,'b',tilt,respx,'r');
set(gca,'xlim',[-90,90],'xtick',[-90:15:90])
legend('Co-polarization','Cross-polarization');
title('Signatures for linear polarization');
xlabel('Tilt angle (degrees)');
ylabel('Signature');
```

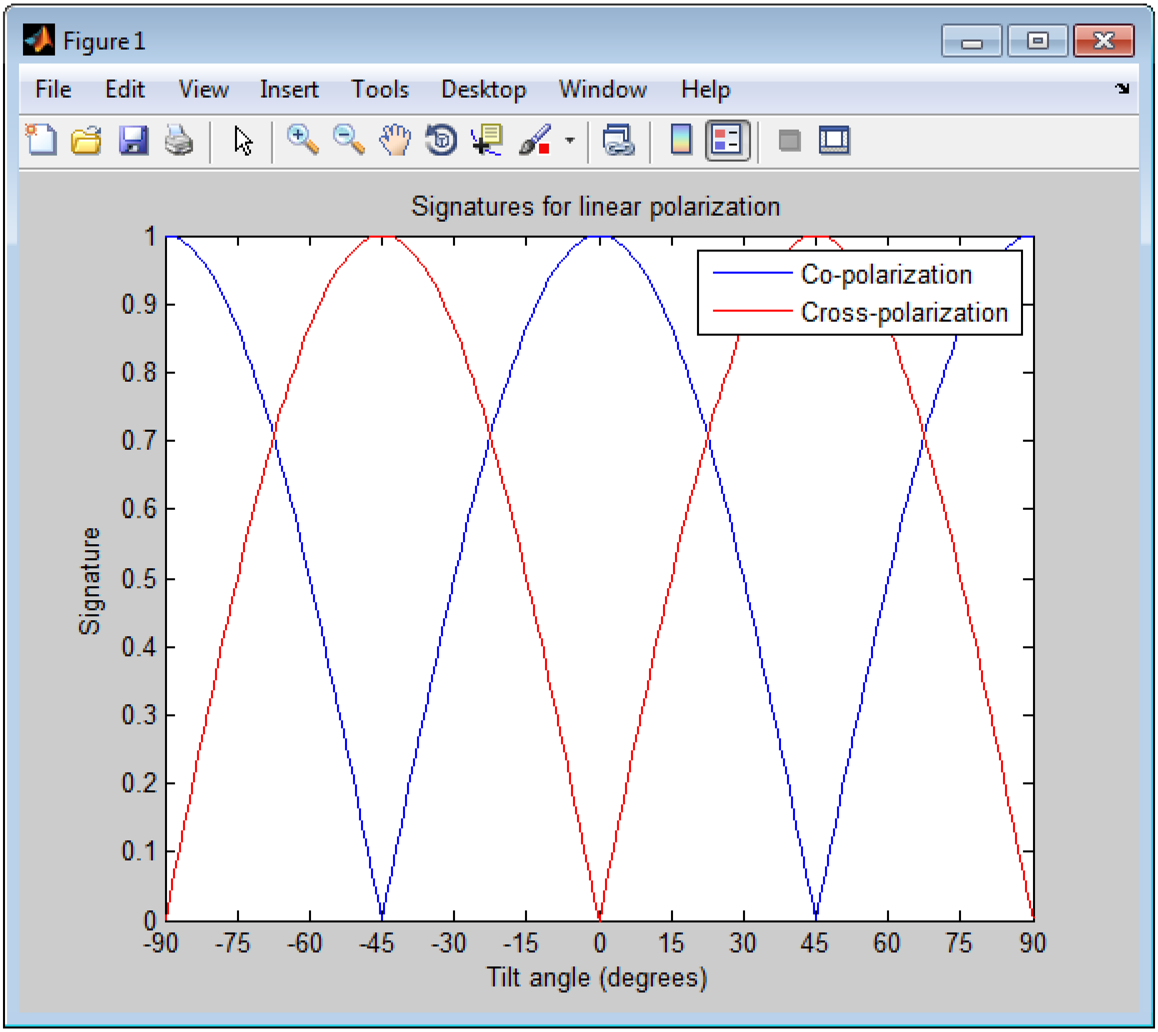


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Signatures for linear polarization
Co-polarization
Cross-polarization
Signature
Tilt angle (degrees)
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-90
-75
-60
-45
-30
-15
0
15
30
45
60
75
90

Copolarization Signature of Dihedral for Right and Left Circular Polarization

Specify right and left circular polarizations. The RCSM for a dihedral is diagonal.

```
rscmat = [-1,0;0,1];
el = [-45, 45];
tilt = 0;
respc = polsignature(rscmat,'c',el,tilt);
respx = polsignature(rscmat,'x',el,tilt);
```

```
respc =

   1   1
respx =

   1   1
```

The responses of the dihedral are the same for each polarization.

Surface Plot of Copolarization Signature of a More General Target

Use a general RCSM matrix to create a 3-D surface plot.

```
rscmat = [1i*2,0.5; 0.5, -1i];
el = [-45:45];
tilt = [-90:90];
polsignature(rscmat,'c',el,tilt);
```

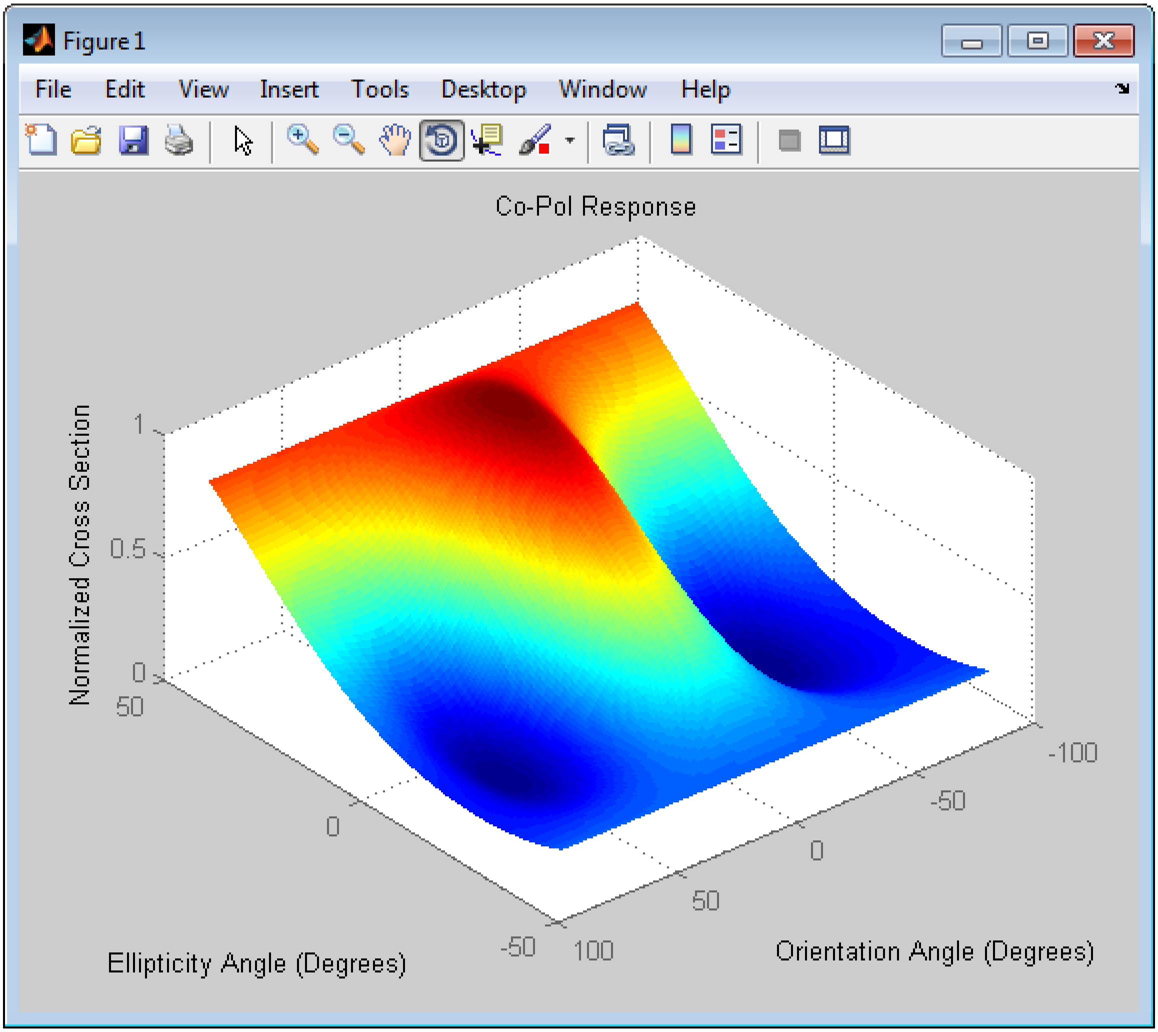


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Co-Pol Response
Normalized Cross Section
1
0.5
0
50
0
-50
100
50
0
-50
-100
Ellipticity Angle (Degrees)
Orientation Angle (Degrees)

Definitions

Scattering Cross-Section Matrix

Scattering cross-section matrix determines response of an object to incident polarized electromagnetic field.

When a polarized plane wave is incident on an object, the amplitude and polarization of the scattered wave may change with respect to the incident wave polarization. The polarization may depend upon the direction in which the scattered wave is observed. The exact way that the polarization changes depends upon the properties of the scattering object. The quantity describing the response of an object to the incident field is called the scattering cross-section matrix, S. The scattering matrix can be measured as follows: when a unit amplitude horizontally polarized wave is scattered, both a horizontal and vertical scattered component are produced. Call these two components S_{HH} and S_{VH}. These are complex numbers containing the amplitude and phase changes from the incident wave. Similarly, when a unit amplitude vertically polarized wave is scattered, the horizontal and vertical scattered component produced are S_{HV} and S_{VV}. Because any incident field can be decomposed into horizontal and vertical components, stack these quantities into a matrix and write the scattered field in terms of the incident field

$$\begin{bmatrix} E_H^{(sc)} \\ E_V^{(sc)} \end{bmatrix} = \begin{bmatrix} S_{HH} & S_{VH} \\ S_{HV} & S_{VV} \end{bmatrix} \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix} = S \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

The scattering cross section matrix depends upon the angles that the incident and scattered fields make with the object. When the incident field is backscattered to the transmitting antenna, the scattering matrix is symmetric.

Polarization Signature

Polarization signature for visualizing scattering cross-section matrix.

To understand how the scattered wave depends upon the polarization of the incident wave, an examination of all possible scattered field polarizations for each incident polarization is required. Because this amount of data is difficult to visualize, you can look at two particular scattered polarizations:

- Choose one polarization that has the same polarization as the incident field (copolarization)

- Choose a second one that is orthogonal to the polarization of the incident field (cross-polarization)

Both the incident and orthogonal polarization states can be specified in terms of the tilt angle-ellipticity angle pair (τ,ε). From the incident field tilt and ellipticity angles, the unit incident polarization vector can be expressed as

$$\begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix} = \begin{bmatrix} \cos\tau & -\sin\tau \\ \sin\tau & \cos\tau \end{bmatrix} \begin{bmatrix} \cos\varepsilon \\ j\sin\varepsilon \end{bmatrix}$$

while the orthogonal polarization vector is

$$\begin{bmatrix} E_H^{(inc)\perp} \\ E_V^{(inc)\perp} \end{bmatrix} = \begin{bmatrix} -\sin\tau & -\cos\tau \\ \cos\tau & -\sin\tau \end{bmatrix} \begin{bmatrix} \cos\varepsilon \\ -j\sin\varepsilon \end{bmatrix}$$

To form the copolarization signature, use the RCSM matrix, *S*, to compute:

$$P^{(co)} = \begin{bmatrix} E_H^{(inc)} & E_V^{(inc)} \end{bmatrix}^* S \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

where `[]*` denotes complex conjugation. For the cross-polarization signature, compute

$$P^{(cross)} = \begin{bmatrix} E_H^{(inc)\perp} & E_V^{(inc)\perp} \end{bmatrix}^* S \begin{bmatrix} E_H^{(inc)} \\ E_V^{(inc)} \end{bmatrix}$$

The output of this function is the absolute value of each signature normalized by its maximum value.

References

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

[2] Fawwaz, U. and C. Elachi. *Radar Polarimetry for Geoscience Applications.* Artech House, 1990.

[3] Lee, J. and E. Pottier. *Polarimetric Radar Imaging: From Basics to Applications.* CRC Press, 2009.

See Also

`polellip` | `polloss` | `stokes`

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');
```

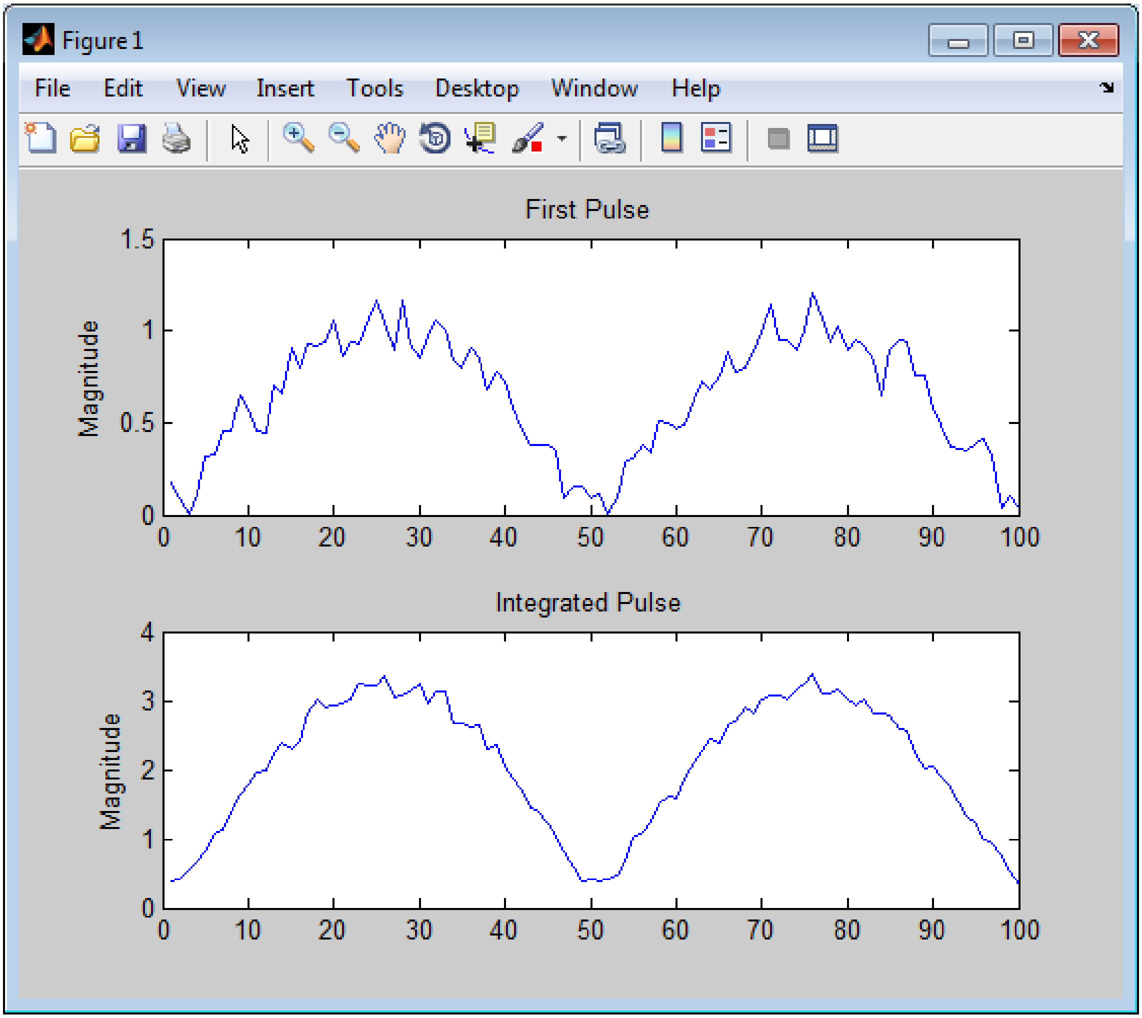


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
First Pulse
Magnitude
1.5
1
0.5
0
0
10
20
30
40
50
60
70
80
90
100
Integrated Pulse
Magnitude
4
3
2
1
0
0
10
20
30
40
50
60
70
80
90
100

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

See Also `phased.MatchedFilter`

radarEquationCalculator

Purpose

Radar equation calculator

Description

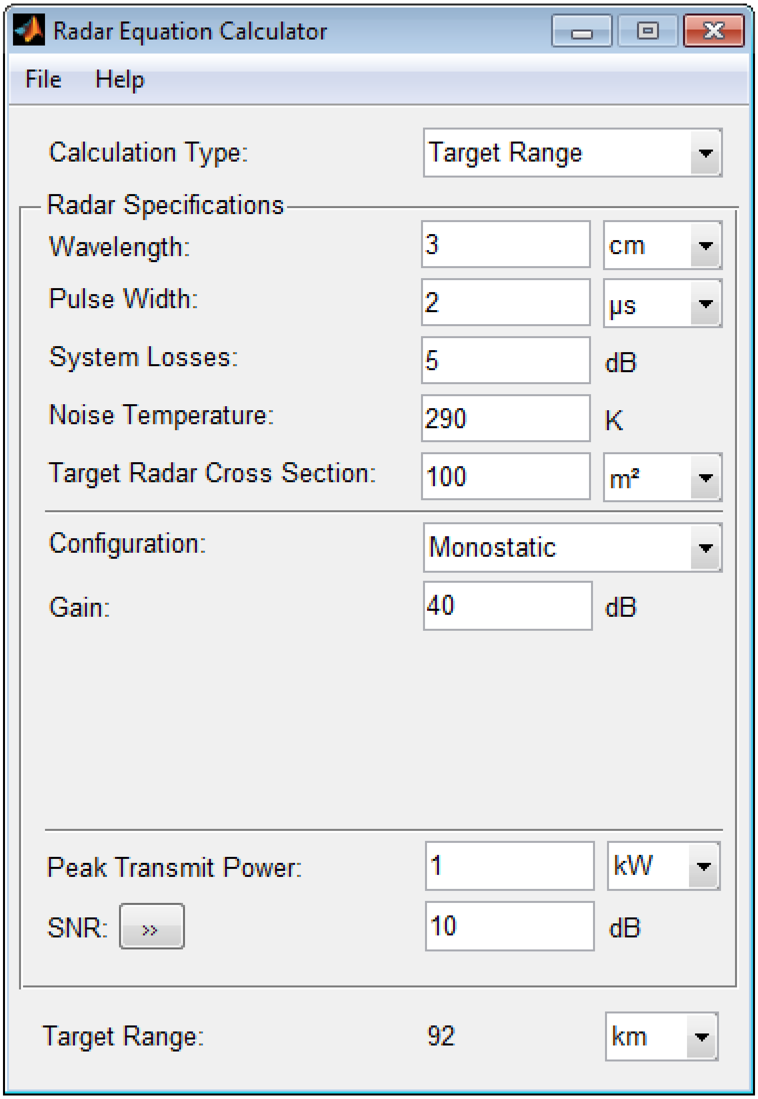
The **Radar Equation Calculator** app is a tool for solving the basic radar equation for monostatic or bistatic radar systems. The radar equation relates target range, transmitted power and received signal SNR. Using this app, you can solve for any one of these three quantities. If you know the transmit power of your radar and the desired received SNR, you can solve for the maximum target range. If you know the target range and desired received SNR, you can compute how much power you need to transmit. Finally, if you know the range and transmit power, you can calculate the received SNR value.

After you choose the type of solution, set other parameters to build a complete model. The principal parameters to specify are target cross-section, wavelength, antenna gains, noise temperature, and overall system losses.

Examples

Maximum Detection Range of a Monostatic Radar

Compute the maximum detection range of a 10 GHz, 1 kW, monostatic radar with a 40 dB antenna gain and a detection threshold of 10 dB. From the **Calculation Type** drop-down list, choose **Target Range** as the solution and choose **Configuration** as `monostatic`. Enter 40 dB for the antenna **Gain**, and set the **Wavelength** to 3 cm. Set the **SNR** detection threshold parameter to 10 dB. Assuming the target is a large airplane, set the **Target Radar Cross Section** value to 100 m^2. Next, specify the **Peak Transmit Power** as 1 kW and the **Pulse Width** as 2 μs. Finally, assume a total of 5 dB **System Losses**.


Radar Equation Calculator
File
Help
»

Calculation Type: Target Range

Radar Specifications

Wavelength: 3 cm

Pulse Width: 2 μs

System Losses: 5 dB

Noise Temperature: 290 K

Target Radar Cross Section: 100 m²

Configuration: Monostatic

Gain: 40 dB

Peak Transmit Power: 1 kW

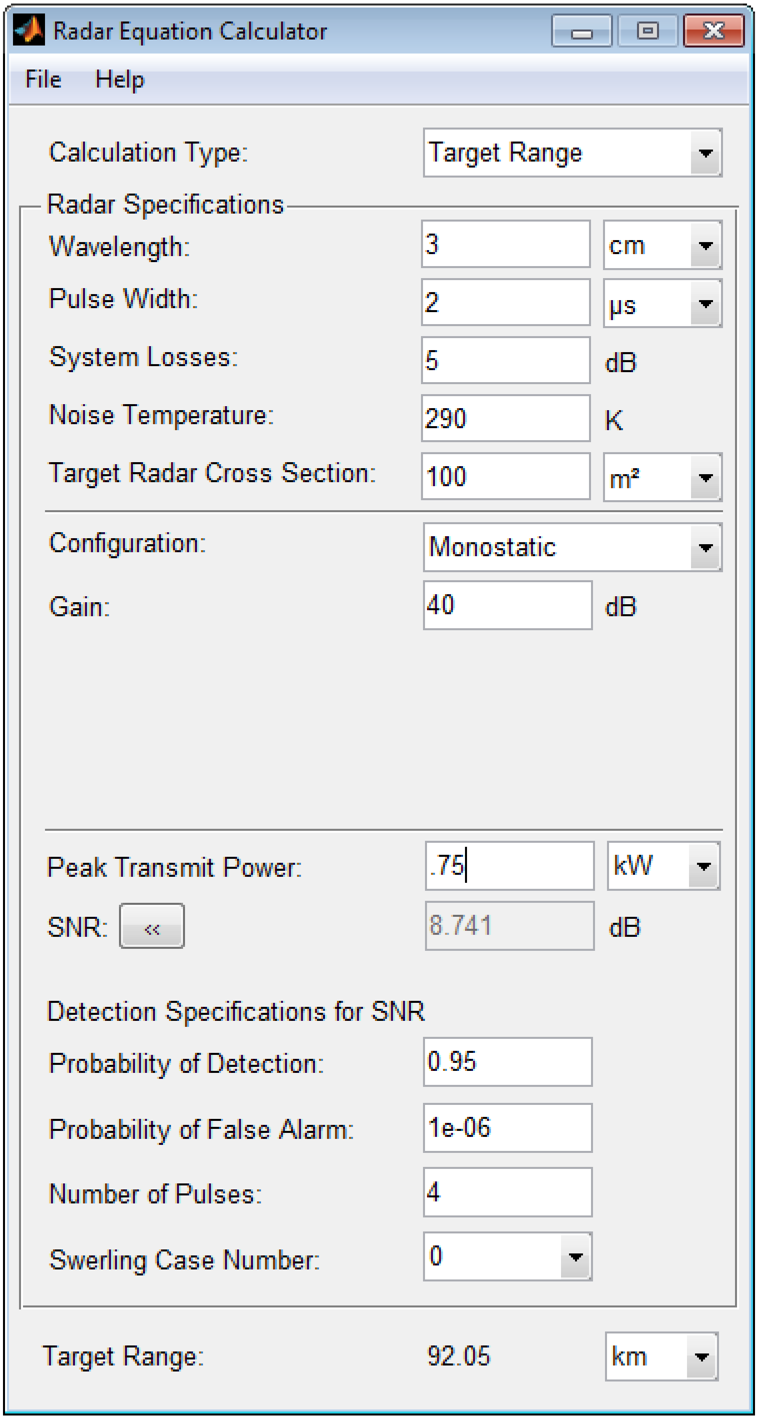
SNR: 10 dB

Target Range: 92 km

The maximum target detection range is 92 km.

Maximum Detection Range of a Monostatic Radar Using Multiple Pulses

Continue with the results from the previous example. Use multiple pulses to reduce the transmitted power while maintaining the same maximum target range. Clicking on the arrows to the right of the **SNR** label opens the **Detection Specifications for SNR** menu. There, set the **Probability of Detection** to 0.95, the **Probability of False Alarm** to 10^{-6}, and the **Number of Pulses** to 4. Then, reduce the **Peak Transmit Power** to 0.75 kW. Assume a nonfluctuating target model, i.e., the **Swerling Case Number** is 0.



Calculation Type: Target Range

Radar Specifications

Wavelength: 3 cm

Pulse Width: 2 μs

System Losses: 5 dB

Noise Temperature: 290 K

Target Radar Cross Section: 100 m²

Configuration: Monostatic

Gain: 40 dB

Peak Transmit Power: .75 kW

SNR: 8.741 dB

Detection Specifications for SNR

Probability of Detection: 0.95

Probability of False Alarm: 1e-06

Number of Pulses: 4

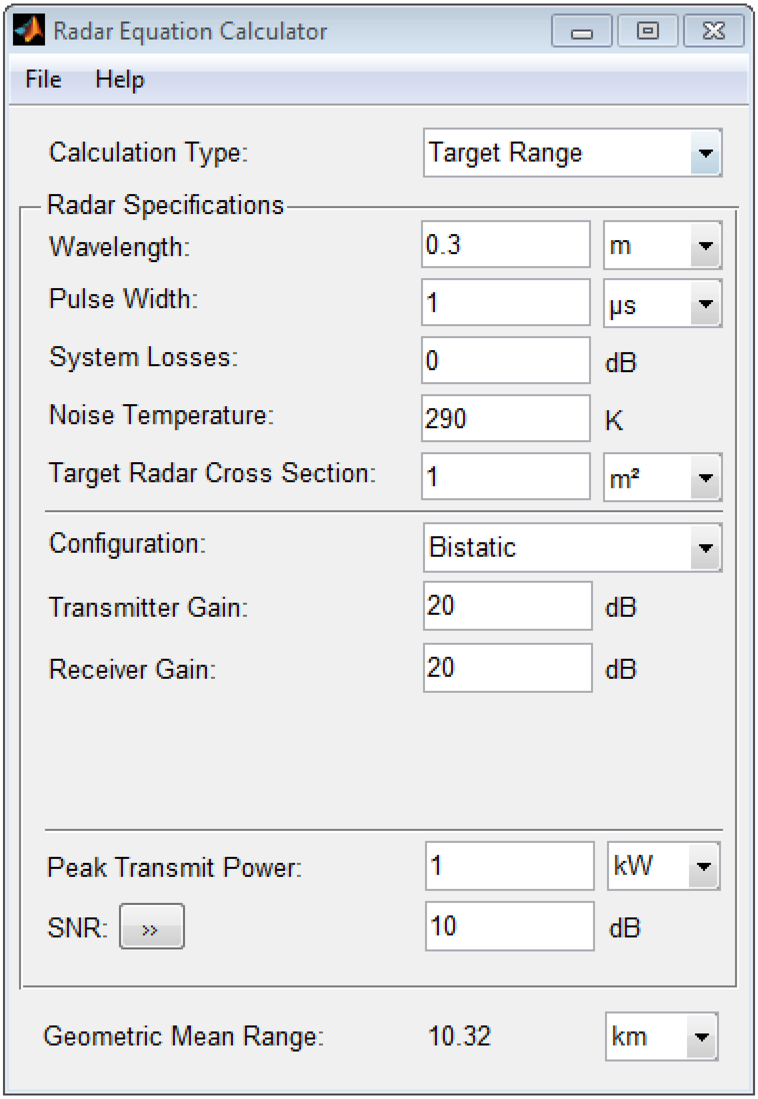
Swerling Case Number: 0

Target Range: 92.05 km

The maximum detection range is approximately the same as in the previous example, but the transmitted power is reduced by 25%.

Maximum Detection Range of Bistatic Radar System

Solve for the geometric mean range of a target for a bistatic radar system. Specify the **Calculation Type** as `Target Range` and **Configuration** as `bistatic`. Next, provide a **Transmitter Gain** and a **Receiver Gain** parameter, instead of the single gain needed in the monostatic case.



Calculation Type: Target Range

Radar Specifications

Wavelength: 0.3 m

Pulse Width: 1 µs

System Losses: 0 dB

Noise Temperature: 290 K

Target Radar Cross Section: 1 m²

Configuration: Bistatic

Transmitter Gain: 20 dB

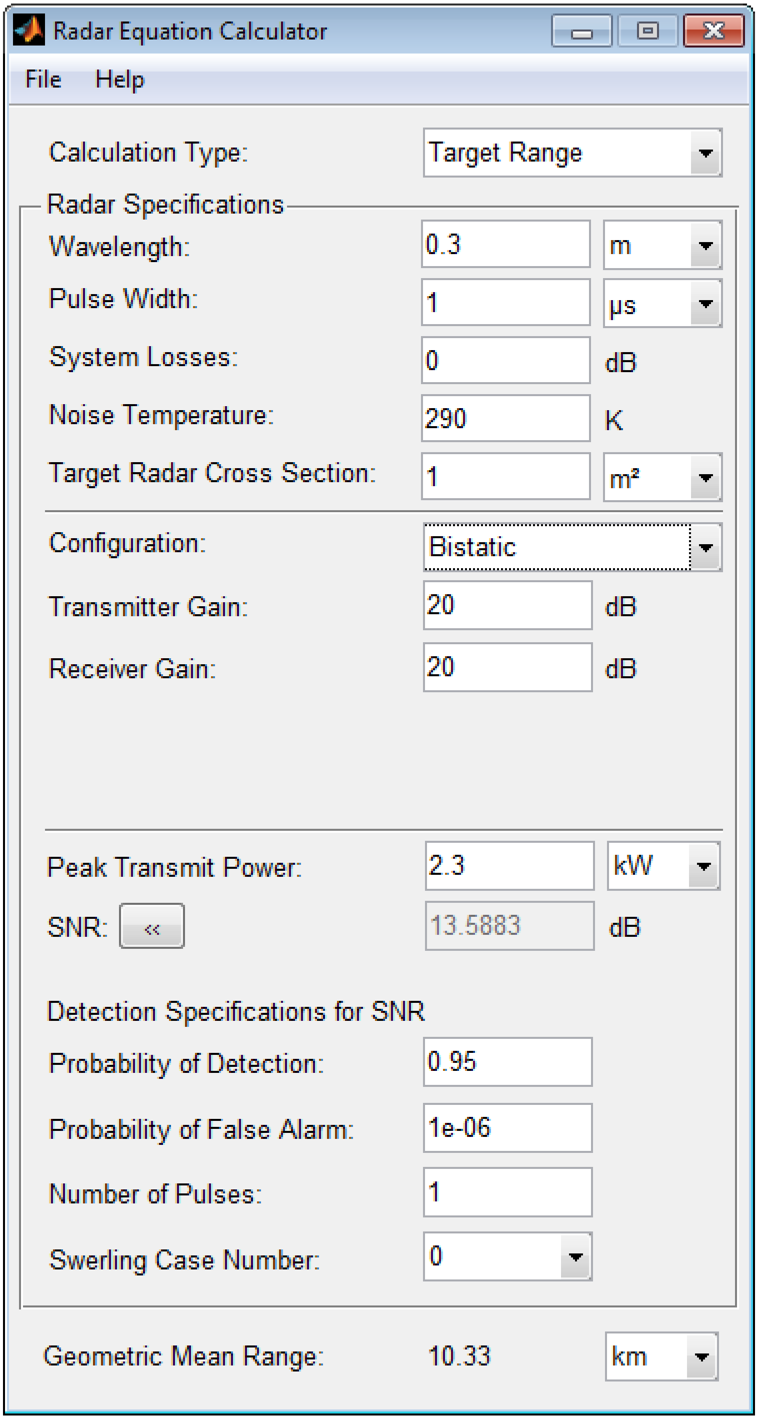
Receiver Gain: 20 dB

Peak Transmit Power: 1 kW

SNR: 10 dB

Geometric Mean Range: 10.32 km

Alternatively, to achieve a particular probability of detection and probability of false alarm, open the **Detection Specifications for SNR** menu. Enter values for **Probability of Detection** and **Probability of False Alarm**, **Number of Pulses**, and **Swerling Case Number**.



Calculation Type: Target Range

Radar Specifications

Wavelength: 0.3 m

Pulse Width: 1 μs

System Losses: 0 dB

Noise Temperature: 290 K

Target Radar Cross Section: 1 m²

Configuration: Bistatic

Transmitter Gain: 20 dB

Receiver Gain: 20 dB

Peak Transmit Power: 2.3 kW

SNR: 13.5883 dB

Detection Specifications for SNR

Probability of Detection: 0.95

Probability of False Alarm: 1e-06

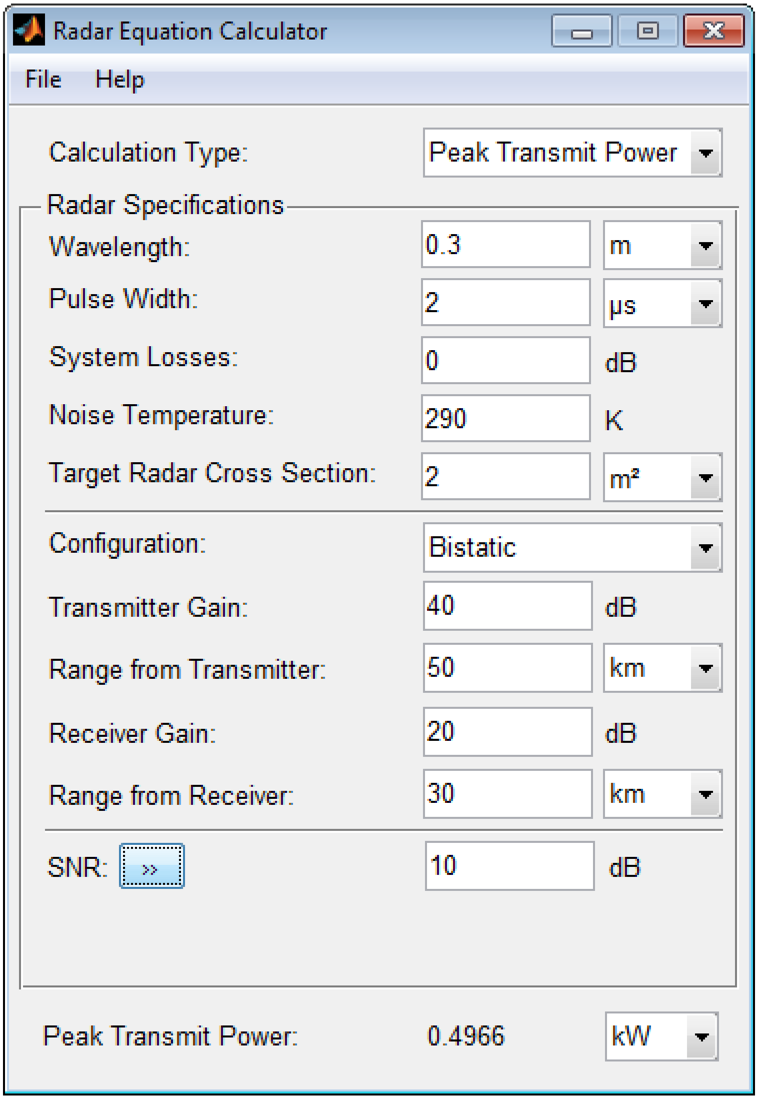
Number of Pulses: 1

Swerling Case Number: 0

Geometric Mean Range: 10.33 km

Required Transmit Power for a Bistatic Radar

Compute the required peak transmit power of a 10 GHz, bistatic X-band radar for a 80 km total bistatic range, and 10 dB received SNR. The system has a 40 dB transmitter gain and a 20 dB receiver gain. The required receiver SNR is 10 dB. From the **Calculation Type** drop-down list, choose **Peak Transmit Power** as the solution type and choose **Configuration** as `bistatic`. From the system specifications, set **Transmitter Gain** to 40 dB and **Receiver Gain** to 20 dB. Set the **SNR** detection threshold to 10 dB and the **Wavelength** to 0.3 m. Assume the target is a fighter aircraft having a **Target Radar Cross Section** value of 2 m^2. Choose **Range from Transmitter** as 50 km, and **Range from Receiver** as 30 km. Finally, set the **Pulse Width** to 2 μs and the **System Losses** to 0 dB.


Radar Equation Calculator
File
Help
Radar Specifications
»

Calculation Type: Peak Transmit Power

Wavelength: 0.3 m

Pulse Width: 2 μs

System Losses: 0 dB

Noise Temperature: 290 K

Target Radar Cross Section: 2 m²

Configuration: Bistatic

Transmitter Gain: 40 dB

Range from Transmitter: 50 km

Receiver Gain: 20 dB

Range from Receiver: 30 km

SNR: 10 dB

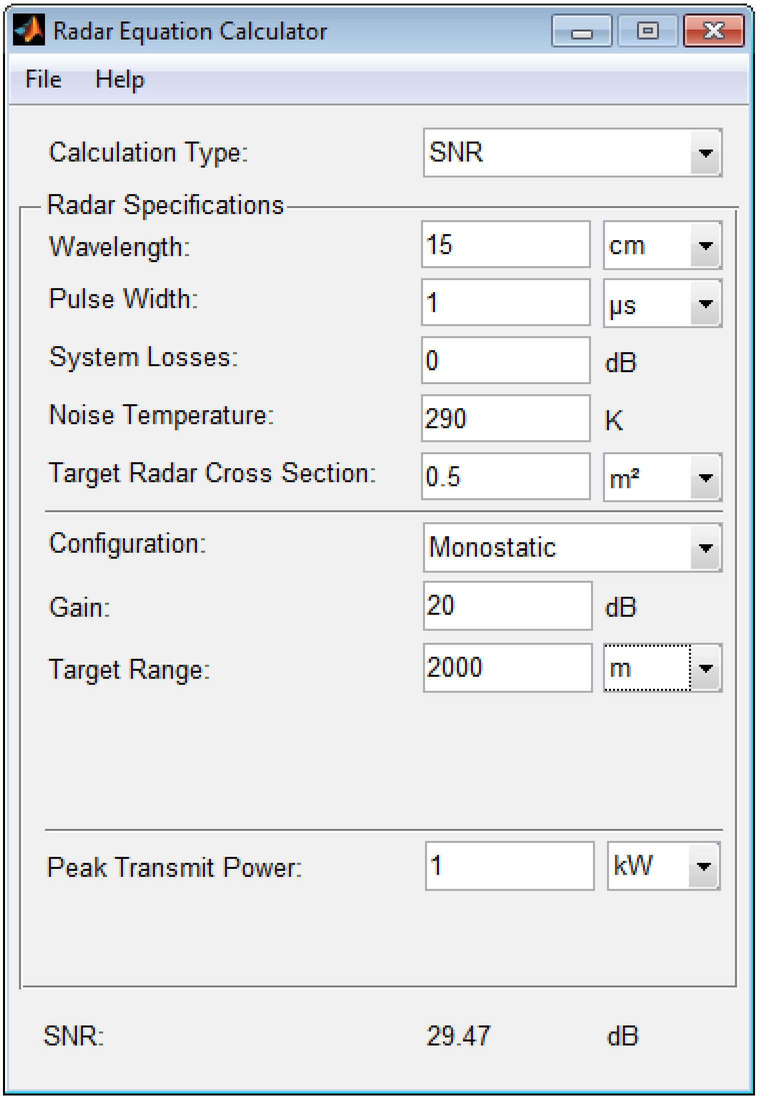
Peak Transmit Power: 0.4966 kW

The required Peak Transmit Power is about 0.5 kW.

Receiver SNR for a Monostatic Radar

Compute the received SNR for a monostatic radar with 1 kW peak transmit power with a target at a range of 2 km. Assume a 2 GHz radar frequency and 20 dB antenna gain. From the **Calculation Type** drop-down list, choose **SNR** as the solution type and set the **Configuration** as `monostatic`. Set the **Gain** to 20, the **Peak Transmit Power** to 1 kW, and the **Target Range** to 2000 m. Set the **Wavelength** to 15 cm.

Find the received SNR of a small boat having a **Target Radar Cross Section** value of 0.5 m^2. The **Pulse Width** is 1 μs and **System Losses** are 0 dB.


Radar Equation Calculator
File
Help
Radar Specifications

Calculation Type: SNR

Wavelength: 15 cm

Pulse Width: 1 μs

System Losses: 0 dB

Noise Temperature: 290 K

Target Radar Cross Section: 0.5 m²

Configuration: Monostatic

Gain: 20 dB

Target Range: 2000 m

Peak Transmit Power: 1 kW

SNR: 29.47 dB

See Also “Radar Equation Theory”

radareqpow

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 2-228 and the output noise power in “Receiver Output Noise Power” on page 2-228, 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 2-234
- Output noise power in "Receiver Output Noise Power" on page 2-235

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

$$R = \left(\frac{N P_t \tau G_t G_r \lambda^2 \sigma}{P_r (4\pi)^3 k T_s L}\right)^{1/4}$$

For bistatic radars, the theoretical maximum detectable range is the geometric mean of the ranges from the target to the transmitter and receiver:

$$\sqrt{R_t R_r} = \left(\frac{N P_t \tau G_t G_r \lambda^2 \sigma}{P_r (4\pi)^3 k T_s L}\right)^{1/4}$$

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. `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 2-241.

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 2-240

- Output noise power in "Receiver Output Noise Power" on page 2-241

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`

Purpose

Vertical coverage diagram

Syntax

```
[vcp,vcpangles] = radarvcd(freq,rfs,anht)
[vcp,vcpangles] = radarvcd( ___ ,Name,Value)

radarvcd( ___ )
```

Description

`[vcp,vcpangles] = radarvcd(freq,rfs,anht)` calculates the vertical coverage pattern of a narrowband radar antenna. The "Vertical Coverage Pattern" on page 2-252 is the radar's range, `vcp`, as a function of elevation angle, `vcpangles`. The vertical coverage pattern depends upon three parameters. These parameters are the radar's maximum free-space detection range, `rfs`, the radar frequency, `freq`, and the antenna height, `anht`.

`[vcp,vcpangles] = radarvcd( ___ ,Name,Value)` allows you to specify additional input parameters as Name-Value pairs. You can specify additional name-value pair arguments in any order as (`Name1`,`Value1`,...,`NameN`,`ValueN`). This syntax can use any of the input arguments in the previous syntax.

`radarvcd( ___ )` displays the vertical coverage diagram for a radar system. The plot is the locus of points ofmaximum radar range as a function of target elevation. This plot is also known as the *Blake chart*. To create this chart, `radarvcd` invokes the function `blakechart` using default parameters. To produce a Blake chart with different parameters, first call `radarvcd` to obtain `vcp` and `vcpangles`. Then, call `blakechart` with user-specified parameters. This syntax can use any of the input arguments in the previous syntaxes.

Input Arguments

freq - Radar frequency

real-valued scalar less than 10 GHz

Radar frequency specified as a real-valued scalar less than 10 GHz (10e9).

Example: 100e6

Data Types
`double`

rfs - Free-space range

real-valued scalar

Free-space range specified as a real-valued scalar. Range units are set by the `RangeUnit` Name-Value pair.

Example: 100e3

Data Types
`double`

anht - Radar antenna height

real-valued scalar

Radar antenna height specified as a real-valued scalar. Height units are set by the `HeightUnit` Name-Value pair.

Example: 10

Data Types
`double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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`.

Example: ‘HeightUnit’, k‘m’

’RangeUnit’ - Radar range units

`'km'` (default) | `'nmi'` | `'mi'` | `'ft'` | `'m'`

Radar range units denoting kilometers, nautical miles, miles, feet or meters. This name-value pair specifies the units for the free-space range argument, `rfs`, and the output vertical coverage pattern, `vcp`.

Example: `'mi'`

Data Types
`char`

'HeightUnit' - Antenna height units

`'m'` (default) | `'nmi'` | `'mi'` | `'km'` | `'ft'`

Antenna height units denoting meters, nautical miles, miles, kilometers, or feet. This name-value pair specifies the units for the antenna height, `anht`, and the `'SurfaceRoughness'` name-value pair.

Example: `'m'`

Data Types
`char`

'Polarization' - Transmitted wave polarization

`` `H' `` (default) | `'H'` | `'V'`

Transmitted wave polarization specified as `'H'` for horizontal polarization and `'V'` for vertical polarization.

Example: `'V'`

Data Types
`char`

'SurfaceDielectric' - Dielectric constant of reflecting surface

frequency dependent model (default) | complex-valued scalar

Dielectric constant of reflecting surface specified as complex-valued scalar. When omitted, the dielectric constant is taken from a frequency-dependent seawater dielectric model derived in Blake[1].

Example: 70

Data Types
`double`

'SurfaceRoughness' - Surface roughness

0 (default) | real-valued scalar

Surface roughness specified as a non-negative real scalar. Surface roughness is a measure of the height variation of the reflecting surface. The roughness is modeled as a sinusoid wave with crest-to-trough height given by this value. A value of 0 indicates a smooth surface. The units for surface roughness height is specified by the value of the `'HeightUnit'` Name-Value pair.

Example: 2

Data Types
`double`

'AntennaPattern' - Antenna elevation pattern

real-valued *N*-by-1 column vector

Antenna elevation pattern, specified as a real-valued *N*-by-1 column vector. Values for `'AntennaPattern'` must be specified together with values for `'PatternAngles'`.

```
ath = linspace(-pi/2, pi/2, 361);
HPBW = 10*pi/180;
k = 1.39157/sin(HPBW/2);
u = k*sin(ath);
apat = sinc(u/pi);
```

Example: `cosd([ 90:90])`

Data Types
`double`

'PatternAngles' - Antenna pattern elevation angles

real-valued *N*-by-1 column vector

Antenna pattern elevation angles specified as a real-valued *N*-by-1 column vector. The size of the vector specified by `'PatternAngles'` must be the same as that specified by `'AntennaPattern'`. Angle units are expressed in degrees and must lie between –90° and 90°. In general, to properly compute the coverage, the antenna pattern should fill the whole range from –90° to 90°.

Example: `[-90:90]`

Data Types
`double`

'TiltAngle' - Antenna tilt angle

real-valued scalar

Antenna tilt angle specified as a real-valued scalar. The tilt angle is the elevation angle of the antenna with respect to the surface. Angle units are expressed in degrees.

Example: 10

Data Types
`double`

'MaxElevation' - Maximum elevation angle

real-valued scalar

Maximum elevation angle, specified as a real-valued scalar. The maximum elevation angle is the largest angle for which the vertical coverage pattern is calculated. Angle units are expressed in degrees.

Example: 70

Data Types
`double`

Output Arguments

vcp - Vertical coverage pattern

real-valued vector

Vertical coverage pattern returned as a real-valued, K-by-1 column vector. The vertical coverage pattern is the actual maximum range of the radar. Each entry of the vertical coverage pattern corresponds to one of the angles returned in `vcpangles`.

vcpangles - Vertical coverage pattern angles

real-valued vector

Vertical coverage pattern angles returned as a *K*-by-1 column vector. The angles range from –90° to 90°.

Examples

Plot Vertical Coverage Pattern Using Default Parameters

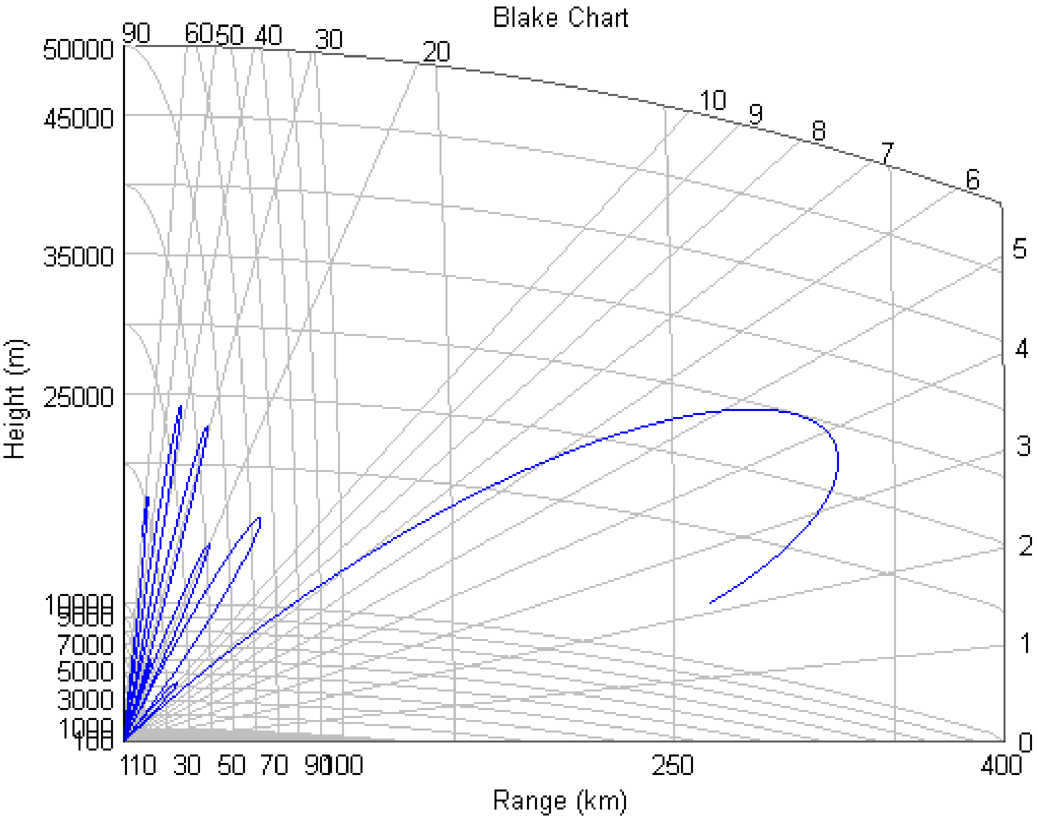
Set the frequency to 100 MHz, the antenna height to 10 m, and the free-space range to 200 km. The antenna pattern, surface roughness, antenna tilt angle, and field polarization assume their default values as specified in the `AntennaPattern`, `SurfaceRoughness`, `TiltAngle`, and `Polarization` properties.

Obtain an array of vertical coverage pattern values and angles.

```
freq = 100e6;
ant_height = 10;
rng_fs = 200;
[vcp,vcpangles] = radarvcd(freq,rng_fs,ant_height);
```

To see the vertical coverage pattern, omit the output arguments.

```
freq = 100e6;
ant_height = 10;
rng_fs = 200;
radarvcd(freq,rng_fs,ant_height);
```



Vertical Coverage Pattern with Specified Antenna Pattern

Set the frequency to 100 MHz, the antenna height to 10 m, and the free-space range to 200 km. The antenna pattern is a sinc function with 45° half-power width. The surface roughness is set to 1 m. The antenna tilt angle is set to 0°, and the field polarization is horizontal.

```
pat_angles = linspace(-90,90,361)';
pat_u = 1.39157/sind(45/2)*sind(pat_angles);
pat = sinc(pat_u/pi);
```

```
freq = 100e6;
ant_height = 10;
rng_fs = 200;
tilt_ang = 0;
[vcp,vcpangles] = radarvcd(freq,rng_fs,ant_height,...
    'RangeUnit','km','HeightUnit','m',...
    'AntennaPattern',pat,...
    'PatternAngles',pat_angles,...
    'TiltAngle',tilt_ang,'SurfaceRoughness',1);
```

Plot Vertical Coverage Diagram For User-Specified Antenna

Plot the range-height-angle curve (Blake Chart) for a radar with a user-specified antenna pattern.

Define a sinc-function antenna pattern with a half-power beamwidth of 90 degrees.

```
pat_angles = linspace(-90,90,361)';
pat_u = 1.39157/sind(90/2)*sind(pat_angles);
pat = sinc(pat_u/pi);
```

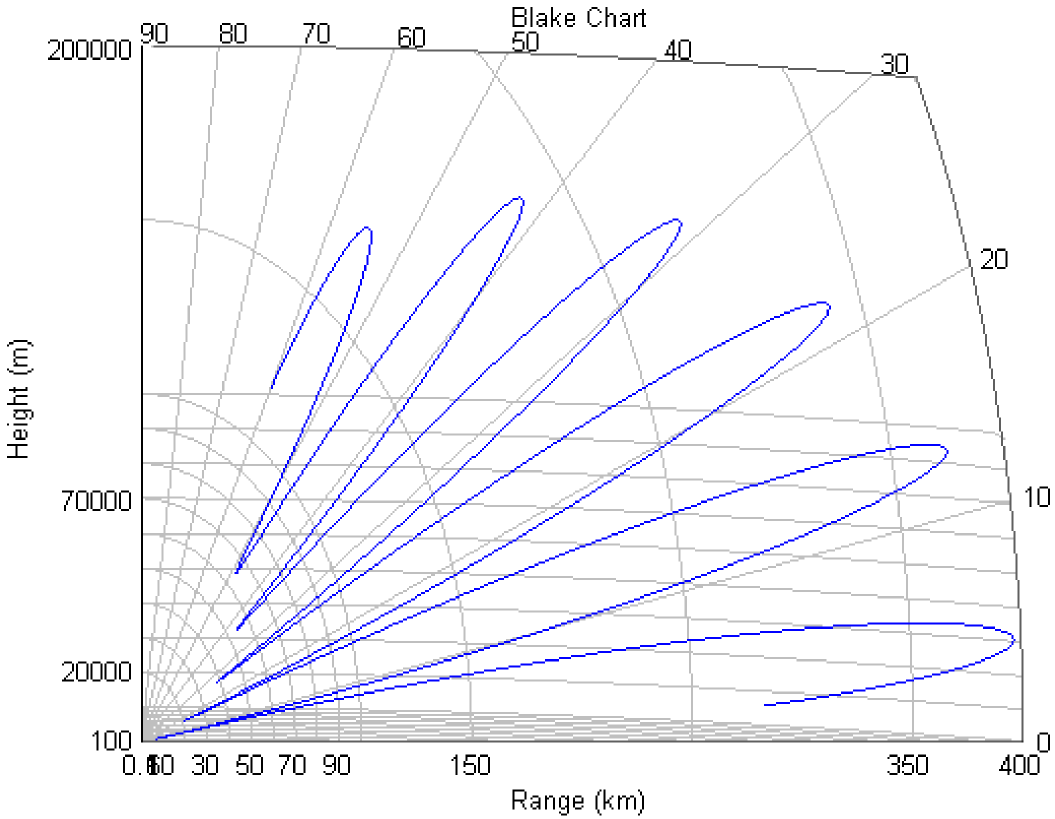
Specify a radar that transmits at 100 MHz. The free-space range is 200 km, the antenna height is 10 meters, the antenna tilt angle is zero degrees, and the surface roughness is one meter.

```
freq = 100e6;
ant_height = 10;
rng_fs = 200;
tilt_ang = 0;
surf_roughness = 1;
```

Create the radar range-height-angle plot.

```
radarvcd(freq,rng_fs,ant_height,...
    'RangeUnit','km','HeightUnit','m',...
    'AntennaPattern',pat,...
    'PatternAngles',pat_angles,...
    'TiltAngle',tilt_ang,...
```

```
          'SurfaceRoughness',surf_roughness);
```



Definitions

Vertical Coverage Pattern

The maximum detection range of a radar antenna can differ, depending on placement. Suppose you place a radar antenna near a reflecting surface, such as the earth's land or sea surface and computed maximum detection range. If you then move the same radar antenna to free space far from any boundaries, a different maximum detection range would result. This is an effect of multi-path interference that occurs when waves, reflected from the surface, constructively add to or nullify the direct path signal from the radar to a target. Multipath interference gives rise to a series of lobes in the vertical plane. The vertical coverage pattern is the plot of the actual maximum detection range of the radar versus target elevation and depends upon the maximum free-space detection range and target elevation angle. See Blake [1].

References

[1] Blake, L.V. *Machine Plotting of Radar Vertical-Plane Coverage Diagrams.* Naval Research Laboratory Report 7098, 1970.

See Also

`blakechart`

radarWaveformAnalyzer

Purpose

Radar waveform analyzer

Description

The **Radar Waveform Analyzer** app is a tool for exploring the properties of various kinds of signals often used in radar and sonar systems. The app lets you determine the basic performance characteristics of the following waveforms:

- Rectangular
- Linear FM
- Stepped FM
- Phase-coded
- FMCW

Each waveform has a set of parameters that are unique to its kind. After you select a signal, the signal parameters menu changes so you can quickly modify the signal. Parameters you can set include the duration, pulse-repetition frequency, number of pulse, bandwidth and sample rate. Changing the propagation speed lets you display properties of sound waves in air and water, or electromagnetic waves. After you enter all the information for a signal of interest, the app displays basic characteristics such as range resolution, Doppler resolution, maximum and minimum range and maximum Doppler.

The **Radar Waveform Analyzer** app lets you produce a variety of plots and images. These are plots of the waveform's

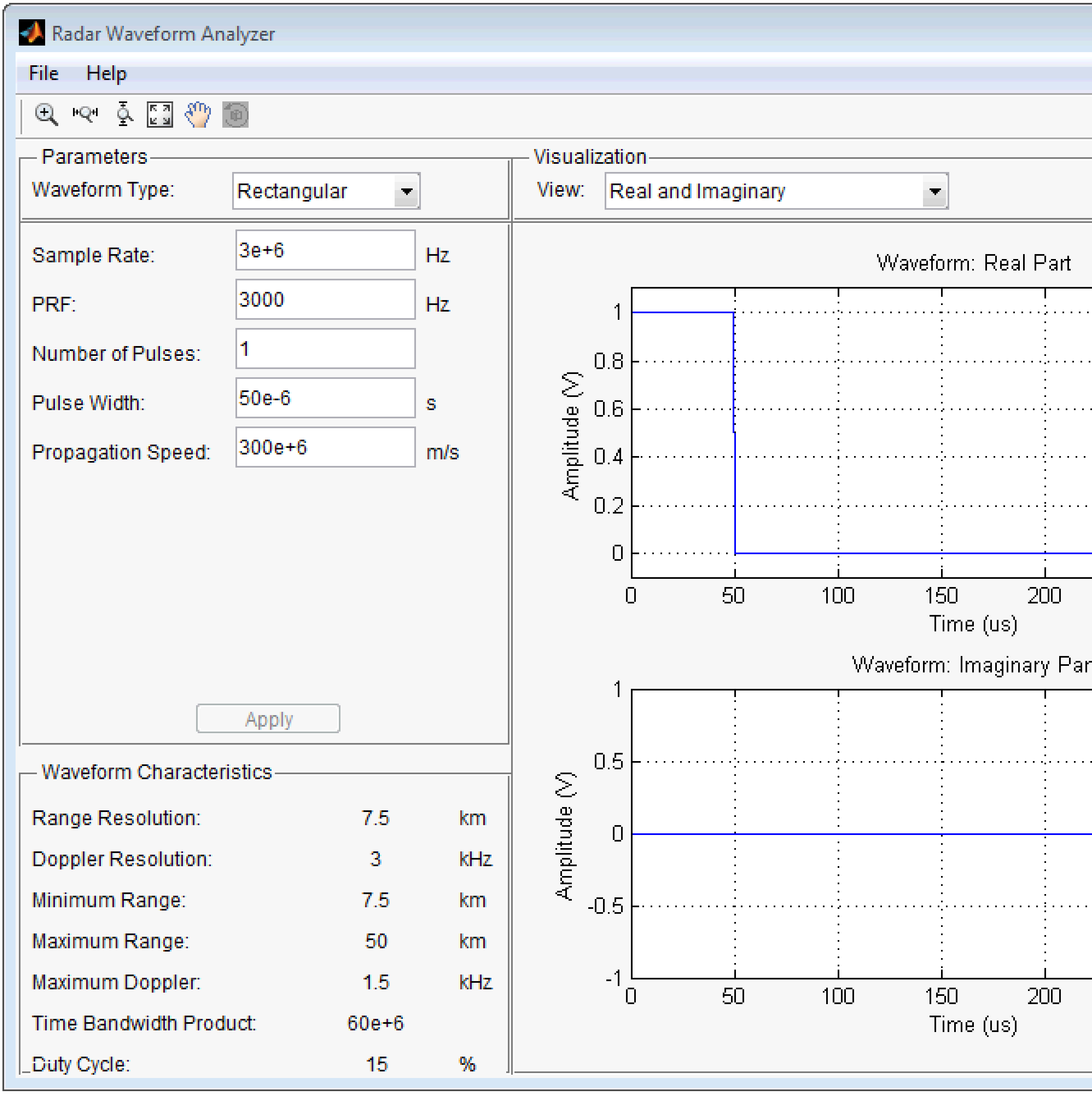
- Real and imaginary components
- Magnitude and phase
- Spectrum
- Spectrogram
- Representations of the ambiguity function
  - Contour
  - Surface
  - Delay cut

  - Doppler cut
- Autocorrelation function

Examples

Rectangular Waveform

Assume a rectangular waveform. Set the **Waveform Type** to `Rectangular`. An ideal rectangular waveform jumps instantaneously to a finite value and stays there for some duration. Assume the radar is designed for a maximum range of 50 km. With this assumption, the propagation time for a signal to go to that range and return is 333 µs. This means you must allow 333 µs between pulses, equivalent to a maximum pulse repetition frequency (**PRF)**) of 3000 Hz. Set the **Pulse Width** to 50 µs. With these values, the app displays a 7.5 km range resolution. The resolution of a rectangular pulse is roughly 1/2 the pulse-width multiplied by the speed of light, which is entered here in the **Propagation Speed** field as 300e6 m/s. The Doppler resolution is approximately the width of the Fourier transform of the pulse. The same analysis can be used for sonar if you assume a much smaller speed of propagation, 1500 m/s. The following figure shows the real and imaginary parts of the waveform. This is the default view on the **View** drop-down list.


Radar Waveform Analyzer
File
Help
Apply
Waveform: Real Part
Amplitude (V)
1
0.8
0.6
0.4
0.2
0
0
50
100
150
200
Time (us)
Waveform: Imaginary Par
Amplitude (V)
1
0.5
0
-0.5
-1
0
50
100
150
200
Time (us)

Parameters

Waveform Type: Rectangular

Sample Rate: 3e+6 Hz

PRF: 3000 Hz

Number of Pulses: 1

Pulse Width: 50e-6 s

Propagation Speed: 300e+6 m/s

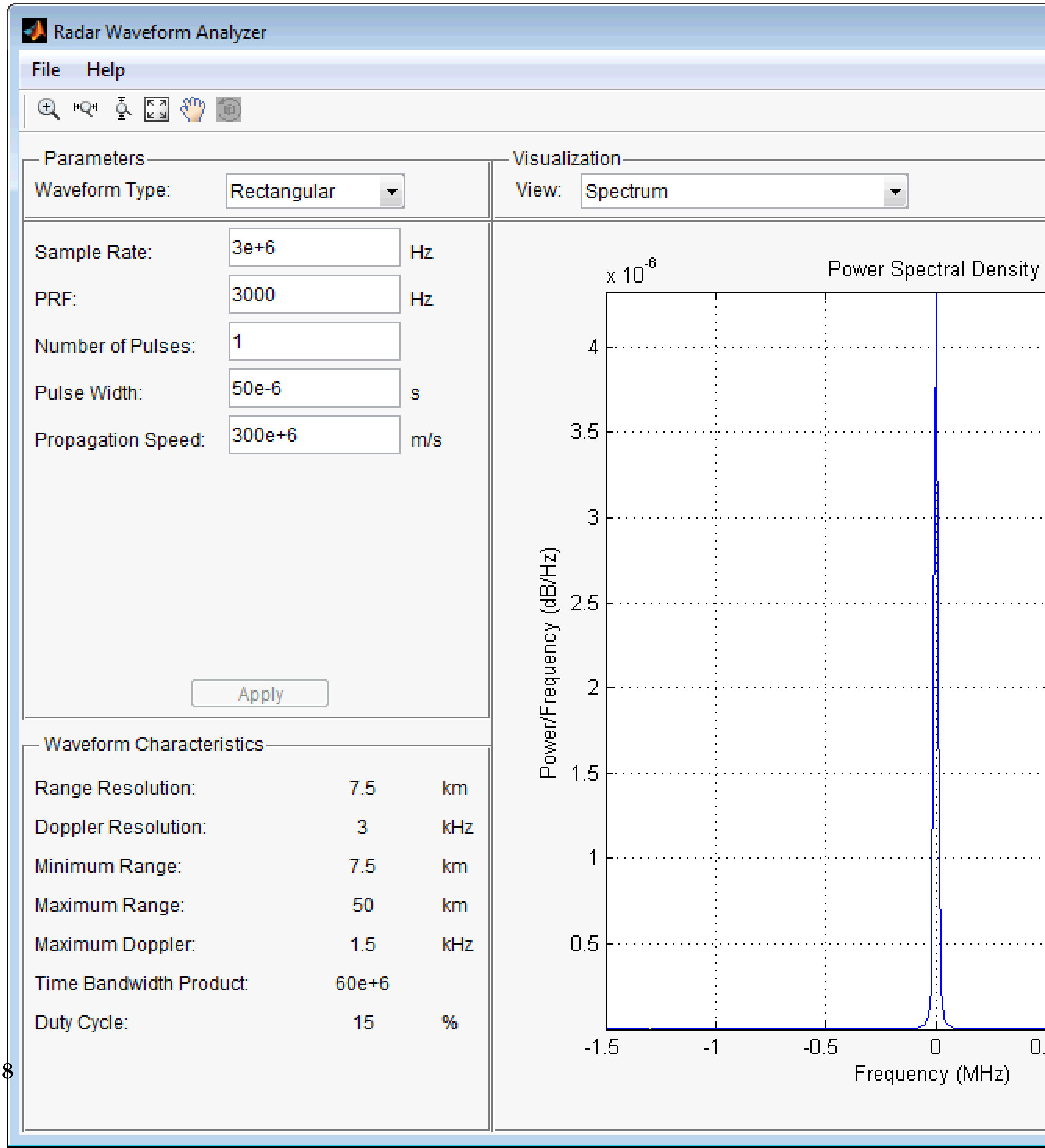
Visualization

View: Real and Imaginary

Waveform Characteristics

Range Resolution:	7.5	km
Doppler Resolution:	3	kHz
Minimum Range:	7.5	km
Maximum Range:	50	km
Maximum Doppler:	1.5	kHz
Time Bandwidth Product:	60e+6	
Duty Cycle:	15	%

Next, you can view the signal spectrum. To do so, select `spectrum` from the **View** drop-down menu.


Radar Waveform Analyzer
File
Help
Apply
Power Spectral Density
x 10^-6
4
3.5
3
2.5
2
1.5
1
0.5
Power/Frequency (dB/Hz)
-1.5
-1
-0.5
0
Frequency (MHz)

Parameters

Waveform Type: Rectangular

Sample Rate: 3e+6 Hz

PRF: 3000 Hz

Number of Pulses: 1

Pulse Width: 50e-6 s

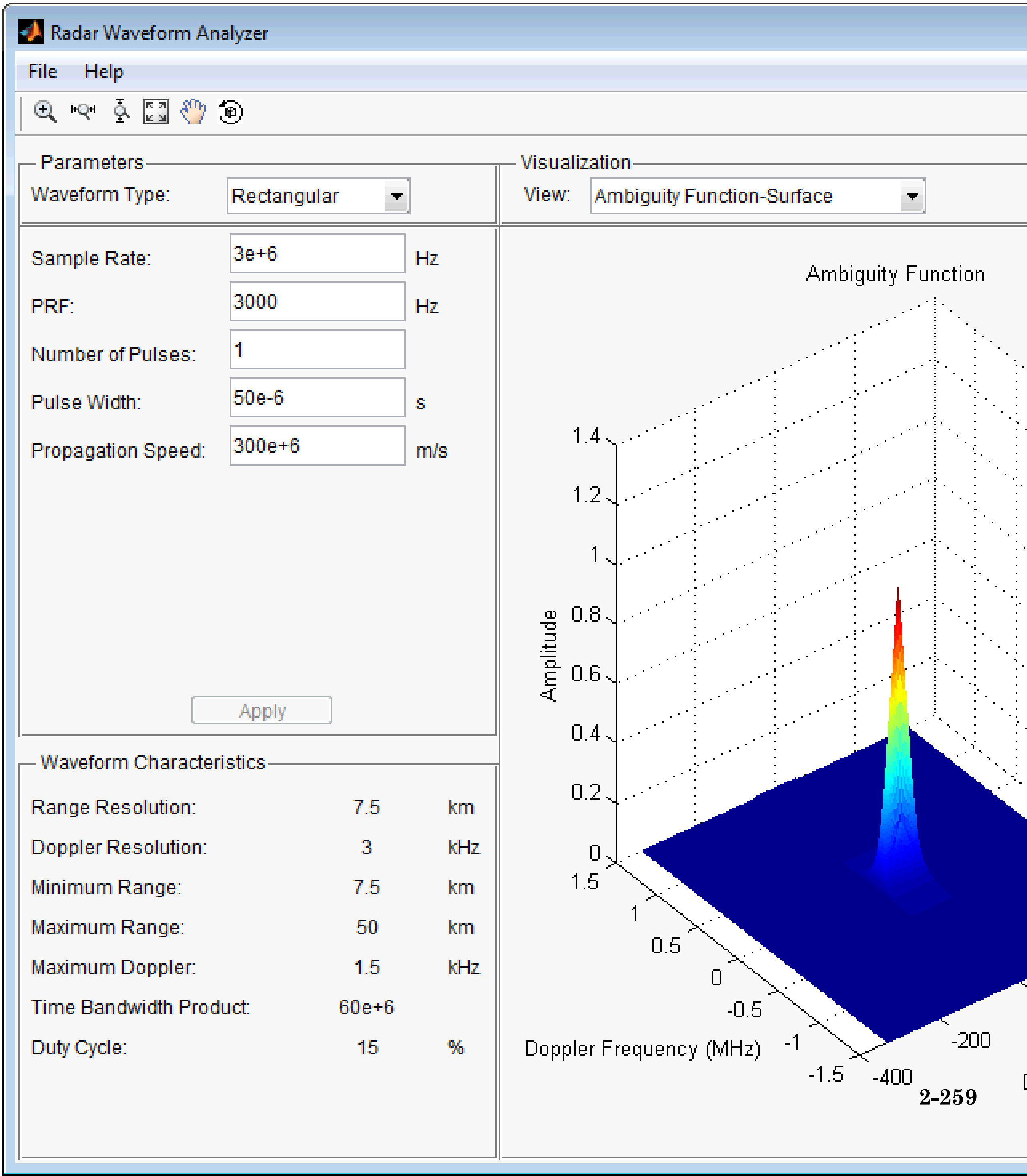
Propagation Speed: 300e+6 m/s

Visualization

View: Spectrum

Waveform Characteristics

Range Resolution:	7.5	km
Doppler Resolution:	3	kHz
Minimum Range:	7.5	km
Maximum Range:	50	km
Maximum Doppler:	1.5	kHz
Time Bandwidth Product:	60e+6	
Duty Cycle:	15	%


Radar Waveform Analyzer
File
Help
Apply
Ambiguity Function
Amplitude
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1.5
1
0.5
0
-0.5
-1
-1.5
Doppler Frequency (MHz)
-400
-200

Parameters

Waveform Type: Rectangular

Sample Rate: 3e+6 Hz

PRF: 3000 Hz

Number of Pulses: 1

Pulse Width: 50e-6 s

Propagation Speed: 300e+6 m/s

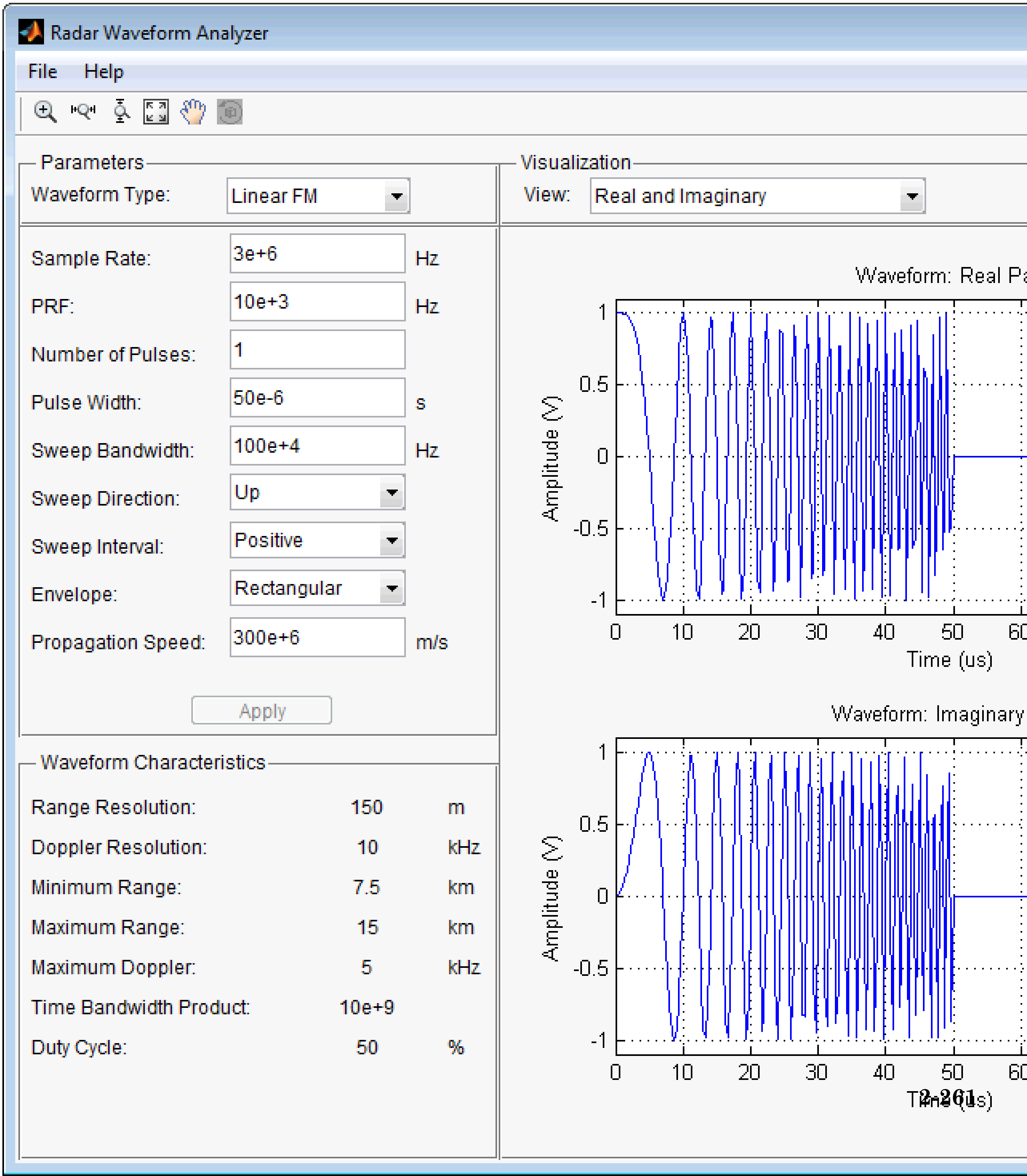
Visualization

View: Ambiguity Function-Surface

Waveform Characteristics

Range Resolution:	7.5	km
Doppler Resolution:	3	kHz
Minimum Range:	7.5	km
Maximum Range:	50	km
Maximum Doppler:	1.5	kHz
Time Bandwidth Product:	60e+6	
Duty Cycle:	15	%

Do this by setting the **Waveform Type** to `Linear FM`. This pulse has a variable frequency which can either increase or decrease as a linear function of time. Choose the **Sweep Direction** as `Up`, and the **Sweep Bandwidth** as 1 MHz. You can see that keeping the same pulse width as before improves the range resolution to 150 m, as shown in the following figure.



Radar Waveform Analyzer

Parameters

Waveform Type: Linear FM

Sample Rate: 3e+6 Hz

PRF: 10e+3 Hz

Number of Pulses: 1

Pulse Width: 50e-6 s

Sweep Bandwidth: 100e+4 Hz

Sweep Direction: Up

Sweep Interval: Positive

Envelope: Rectangular

Propagation Speed: 300e+6 m/s

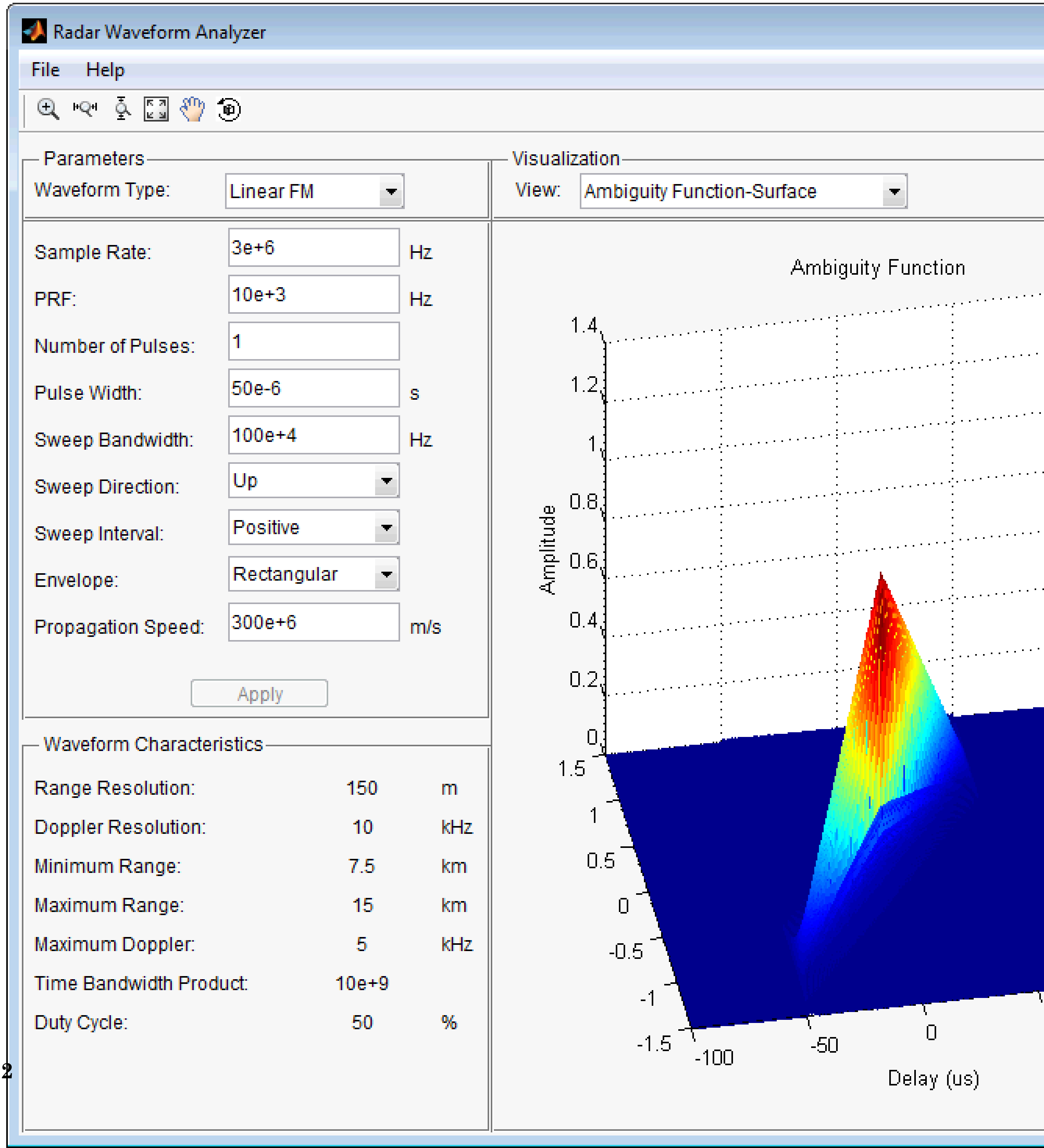
Waveform Characteristics

Range Resolution:	150	m
Doppler Resolution:	10	kHz
Minimum Range:	7.5	km
Maximum Range:	15	km
Maximum Doppler:	5	kHz
Time Bandwidth Product:	10e+9	
Duty Cycle:	50	%

Visualization

View: Real and Imaginary

File
Help
Apply
Waveform: Real Pa
Amplitude (V)
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
Time (us)
Waveform: Imaginary
Amplitude (V)
1
0.5
0
-0.5
-1
0
10
20
30
40
50
60
Time (us)



Radar Waveform Analyzer

File Help

Parameters

Waveform Type: Linear FM

Sample Rate: 3e+6 Hz

PRF: 10e+3 Hz

Number of Pulses: 1

Pulse Width: 50e-6 s

Sweep Bandwidth: 100e+4 Hz

Sweep Direction: Up

Sweep Interval: Positive

Envelope: Rectangular

Propagation Speed: 300e+6 m/s

Apply

Waveform Characteristics

Range Resolution:	150	m
Doppler Resolution:	10	kHz
Minimum Range:	7.5	km
Maximum Range:	15	km
Maximum Doppler:	5	kHz
Time Bandwidth Product:	10e+9	
Duty Cycle:	50	%

Visualization

View: Ambiguity Function-Surface

Ambiguity Function
Amplitude
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1.5
1
0.5
0
-0.5
-1
-1.5
-100
-50
0
Delay (us)

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”

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

range2bw

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` | `range2time` | `phased.FMCWWaveform`

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

range2time

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

rangeangle

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”

rdcoupling

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

real scalar

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` | `stretchfreq2rng` | `phased.FMCWWaveform` | `phased.LinearFMWaveform`

Related Examples

- Automotive Adaptive Cruise Control Using FMCW Technology

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. `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:

$$P_D = \frac{1}{2}\operatorname{erfc}(\operatorname{erfc}^{-1}(2P_{FA}) - \sqrt{\chi})$$

where `erfc` and `erfc`$^{-1}$ are the complementary error function and that function's inverse, and χ is the SNR not expressed in decibels.

For details about the other supported signal types, see [1].

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.

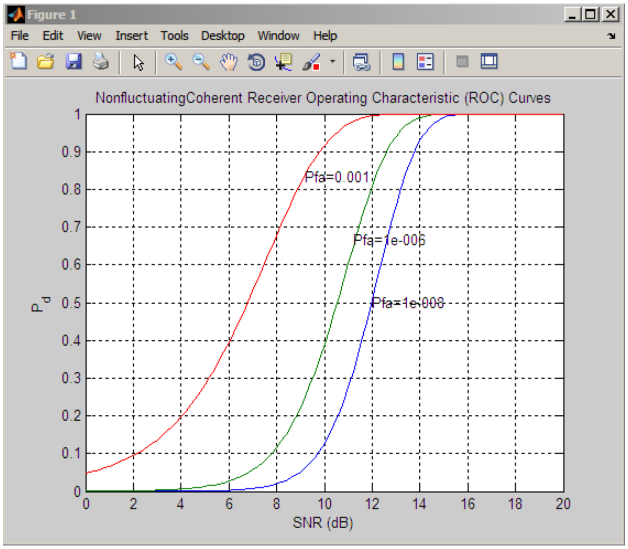
SNR

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. `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 `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.

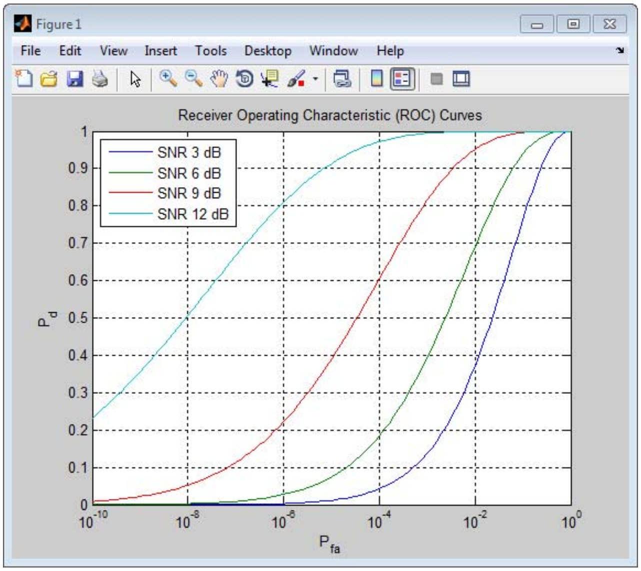
Pfa

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

Direction of arrival using Root MUSIC

Syntax

```
ang = rootmusicdoa(R,nsig)
ang = rootmusicdoa(___,'Name','Value')
```

Description

`ang = rootmusicdoa(R,nsig)` estimates the directions of arrival, `ang`, of a set of plane waves received on a uniform line array (ULA). The estimation uses the *root MUSIC* algorithm. The input arguments are the estimated spatial covariance matrix between sensor elements, `R`, and the number of arriving signals, `nsig`. In this syntax, sensor elements are spaced one-half wavelength apart.

`ang = rootmusicdoa(___,'Name','Value')` allows you to specify additional input parameters in the form of Name-Value pairs. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

R - Spatial covariance matrix

Complex-valued positive-definite N-by-N matrix

Spatial covariance matrix, specified as a complex-valued, positive-definite, N-by-N matrix. In this matrix, N represents the number of elements in the ULA array. If `R` is not Hermitian, a Hermitian matrix is formed by averaging the matrix and its conjugate transpose, `(R+R')/2`.

Example: [4.3162, –0.2777 –0.2337i; –0.2777 + 0.2337i , 4.3162]

Data Types
`double`
Complex Number Support: Yes

nsig - Number of arriving signals

Positive integer

Number of arriving signals, specified as a positive integer.

Example: 2

Data Types
`double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `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`.

Example: ‘ElementSpacing’, 0.4

’ElementSpacing’ - ULA element spacing

0.5 (default) | Real-valued positive scalar

ULA element spacing, specified as a real-valued, positive scalar. Position units are measured in terms of signal wavelength.

Example: 0.4

Data Types
`double`

Output Arguments

ang - Directions of arrival angles

Real-valued 1-by-M row vector

Directions of arrival angle, returned as a real-valued, 1-by-M vector. The dimension M is the number of arriving signals specified in the argument `nsig`. Angle units are degrees and angle values lie between –90° and 90°.

Examples

Three Signals Arriving at Half-Wavelength-Spaced ULA

Assume a half-wavelength spaced uniform line array with 10 elements. Three plane waves arrive from the 0°, –25°, and 30° azimuth directions. Elevation angles are 0°. The noise is spatially and temporally white Gaussian noise.

Set the SNR for each signal to 5 dB. Find the arrival angles.

```
N = 10;
```

```
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25 30];
Nsig = 3;
R = sensorcov(elementPos,angles,db2pow(-5));
doa = rootmusicdoa(R,Nsig)
```

```
doa =

   -0.0000   30.0000  -25.0000
```

The `rootmusicdoa` function finds the correct angles.

Three Signals Arriving at 0.4-Wavelength-Spaced ULA

Assume a uniform line array 10 elements, as in the previous example. But now the element spacing is smaller than one-half wavelength. Three plane waves arrive from the 0°, –25°, and 30° azimuth directions. Elevation angles are 0°. The noise is spatially and temporally white Gaussian noise. The SNR for each signal is 5 dB.

Set the `ElementSpacing` property value to the interelement spacing, 0.4 wavelengths. Find the arrival angles.

```
N = 10;
d = 0.4;
elementPos = (0:N-1)*d;
angles = [0 -25 30];
Nsig = 3;
R = sensorcov(elementPos,angles,db2pow(-5));
doa = rootmusicdoa(R,Nsig,'ElementSpacing',d)
```

```
doa =

  -25.0000    0.0000   30.0000
```

The `rootmusicdoa` function finds the correct angles.

References

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

See Also

`aictest` | `espritdoa` | `rootmusicdoa` | `spsmooth` | `phased.RootMUSICEstimator`

Purpose

Rotation matrix for rotations around x-axis

Syntax

`R = rotx(ang)`

Description

`R = rotx(ang)` creates a 3-by-3 matrix used to rotated a 3-by-1 vector or 3-by-N matrix of vectors around the x-axis by `ang` degrees. When acting on a matrix, each column of the matrix represents a different vector. For the rotation matrix `R` and vector `v`, the rotated vector is given by `R*v`.

Input Arguments

ang - Rotation angle

Real-valued scalar

Rotation angle specified as a real-valued scalar. The rotation angle is positive if the rotation is in the counter-clockwise direction when viewed by an observer looking along the x-axis towards the origin. Angle units are in degrees.

Example: 30.0

Data Types

`double`

Output Arguments

R - Rotation matrix

Real-valued orthogonal matrix

3-by-3 rotation matrix returned as

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

for a rotation angle α.

Examples

Rotation matrix for 30° rotation

Construct the matrix for a rotation of a vector around the x-axis by 30°. Then let the matrix operate on a vector:

```
R = rotx(30)

R =

          1            0            0
          0      0.86603         -0.5
          0          0.5      0.86603

x = [2;-2;4];
y = R*x

y =

          2
    -3.7321
     2.4641
```

Under a rotation around the x-axis, the x-component of a vector is left unchanged.

Definitions

Rotation Matrices

In transforming vectors in three-dimensional space, rotation matrices are often encountered. Rotation matrices are used in two senses: they can be used to rotate a vector into a new position or they can be used to rotate a coordinate basis (or coordinate system) into a new one. In this case, the vector is left alone but its components in the new basis will be different from those in the original basis. In Euclidean space, there are three basic rotations: one each around the x, y and z axes. Each rotation is specified by an angle of rotation. The rotation angle is defined to be positive for a rotation that is counterclockwise when viewed by an observer looking along the rotation axis towards the origin. Any arbitrary rotation can be composed of a combination of these three *(Euler's rotation theorem)*. For example, one can rotated a vector using a sequence of three rotations: $\mathbf{v}' = A\mathbf{v} = R_z(\gamma)R_y(\beta)R_x(\alpha)\mathbf{v}$.

The rotation matrices that rotate a vector around the x, y, and z-axes are given by:

- Counterclockwise rotation around x-axis

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

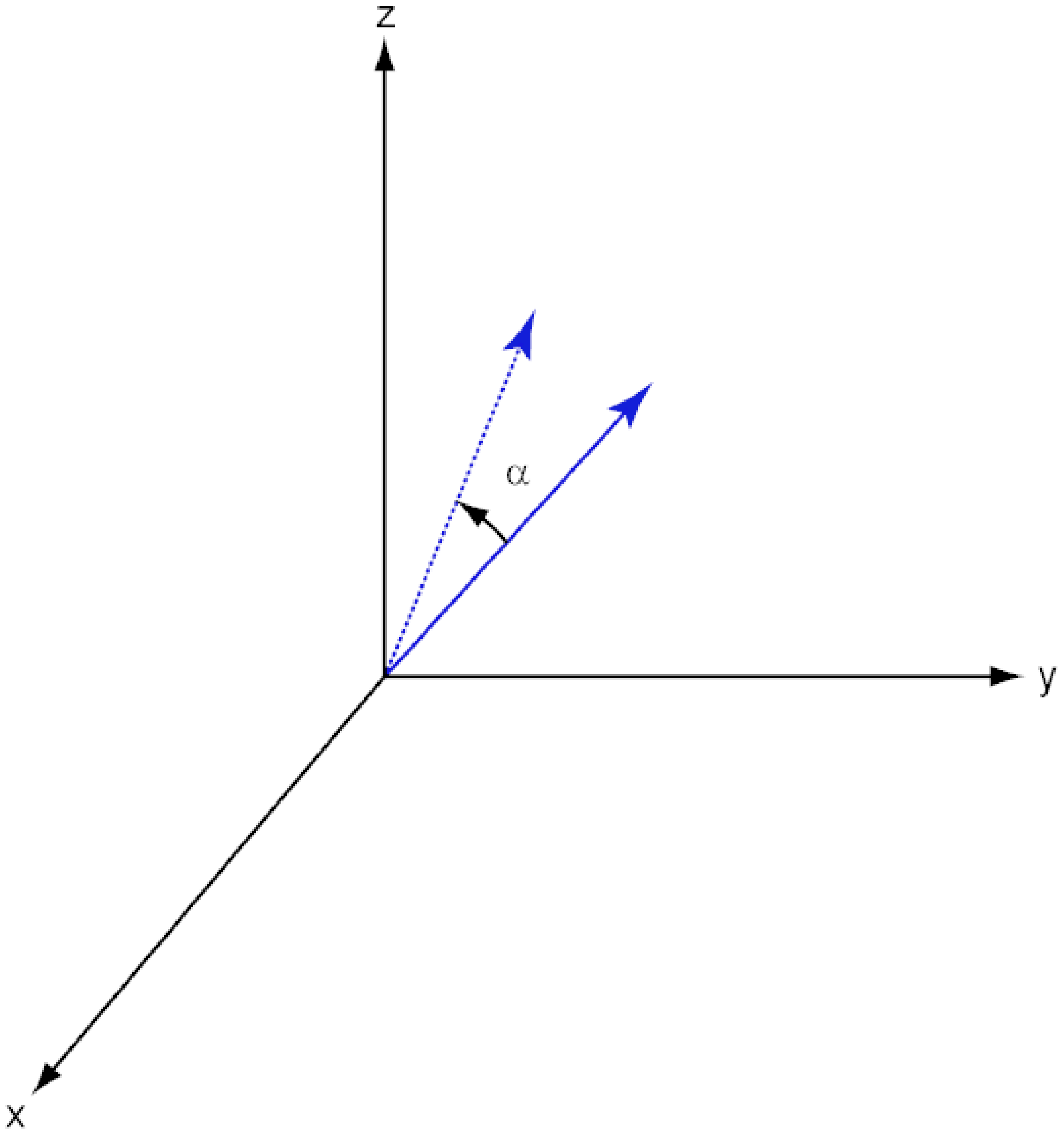
- Counterclockwise rotation around y-axis

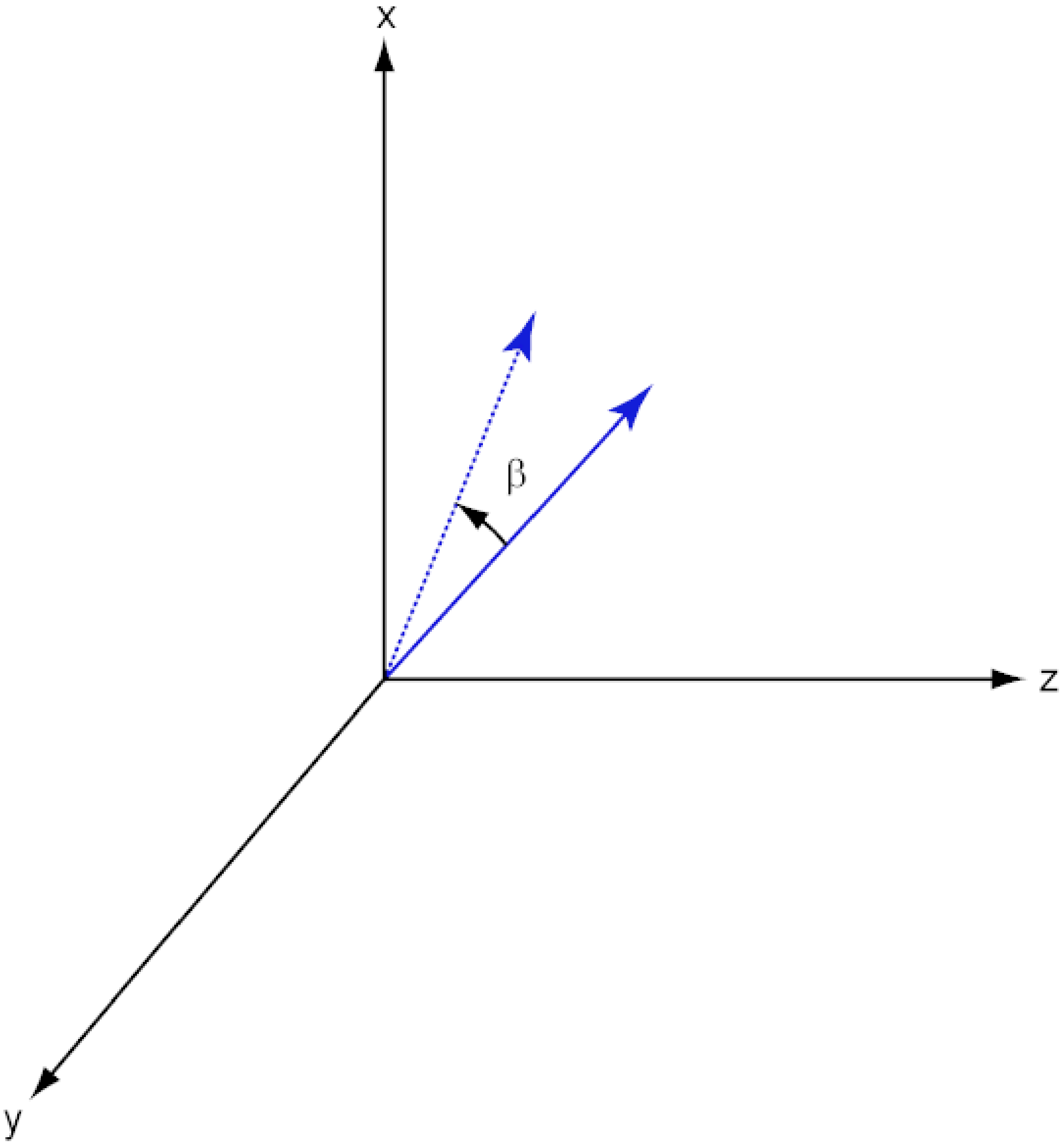
$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

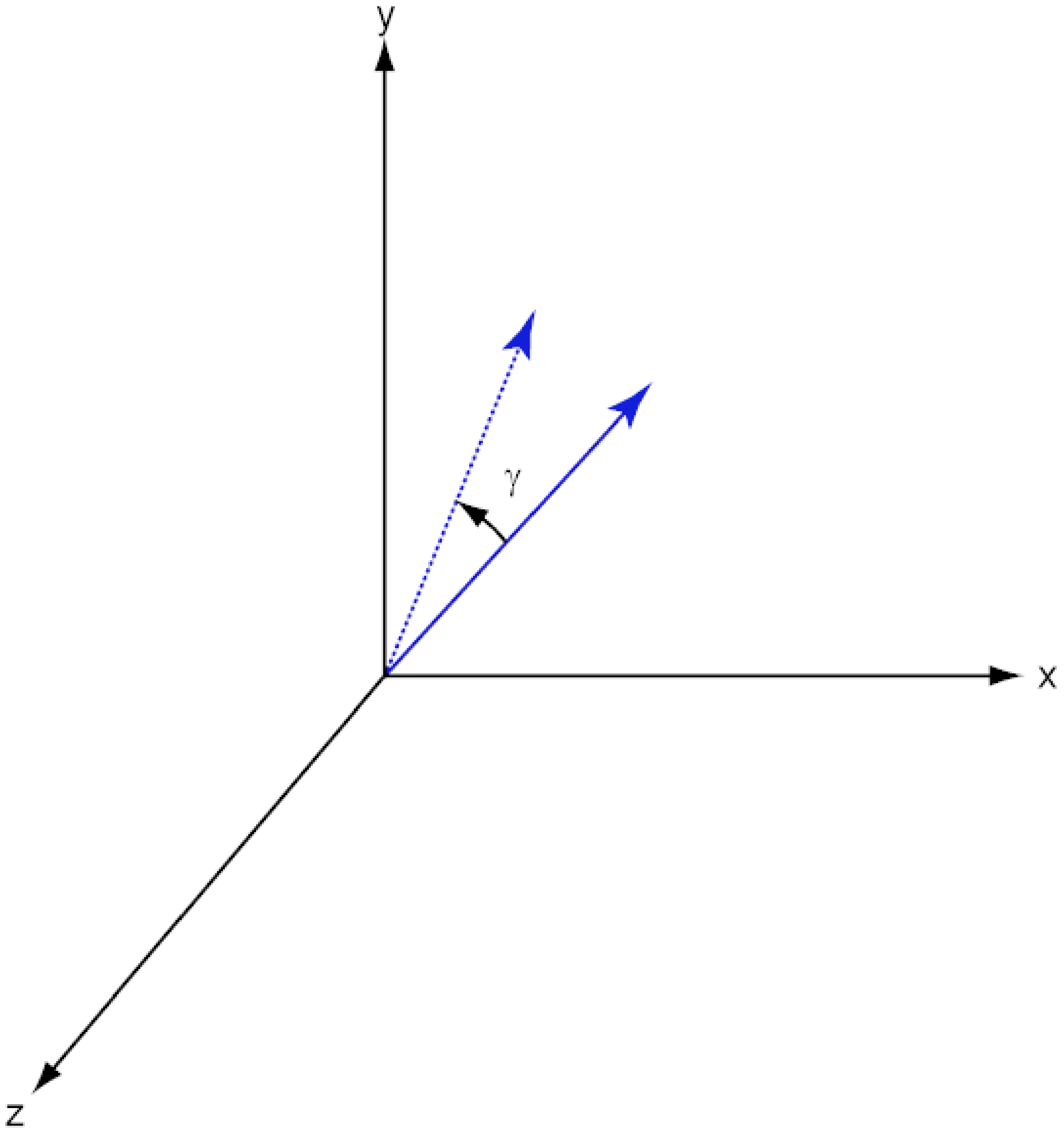
- Counterclockwise rotation around z-axis

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The following three figures show what positive rotations look like for each rotation axis:


z
α
y
x


x
β
z
y


y
γ
x
z

For any rotation, there is an inverse rotation satisfying $A^{-1}A = 1$. For example, the inverse of the x-axis rotation matrix is obtained by changing the sign of the angle:

$$R_x^{-1}(\alpha) = R_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} = R_x^{'}(\alpha)$$

This example illustrates a basic property: the inverse rotation matrix equals the transpose of the original. Rotation matrices satisfy *A'A = 1*, and consequently *det(A) = 1.* Under rotations, vector lengths are preserved as well as the angles between vectors.

We can think of rotations in another way. Consider the original set of basis vectors, $\mathbf{i}, \mathbf{j}, \mathbf{k}$, and rotate them all using the rotation matrix *A*.

This produces a new set of basis vectors $\mathbf{i}^{'}, \mathbf{j}^{'}, \mathbf{k}^{'}$ related to the original by:

$$\begin{aligned} \mathbf{i}' &= A\mathbf{i} \\ \mathbf{j}' &= A\mathbf{j} \\ \mathbf{k}' &= A\mathbf{k} \end{aligned}$$

The new basis vectors can be written as linear combinations of the old ones and involve the transpose:

$$\begin{bmatrix} \mathbf{i}' \\ \mathbf{j}' \\ \mathbf{k}' \end{bmatrix} = A' \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}$$

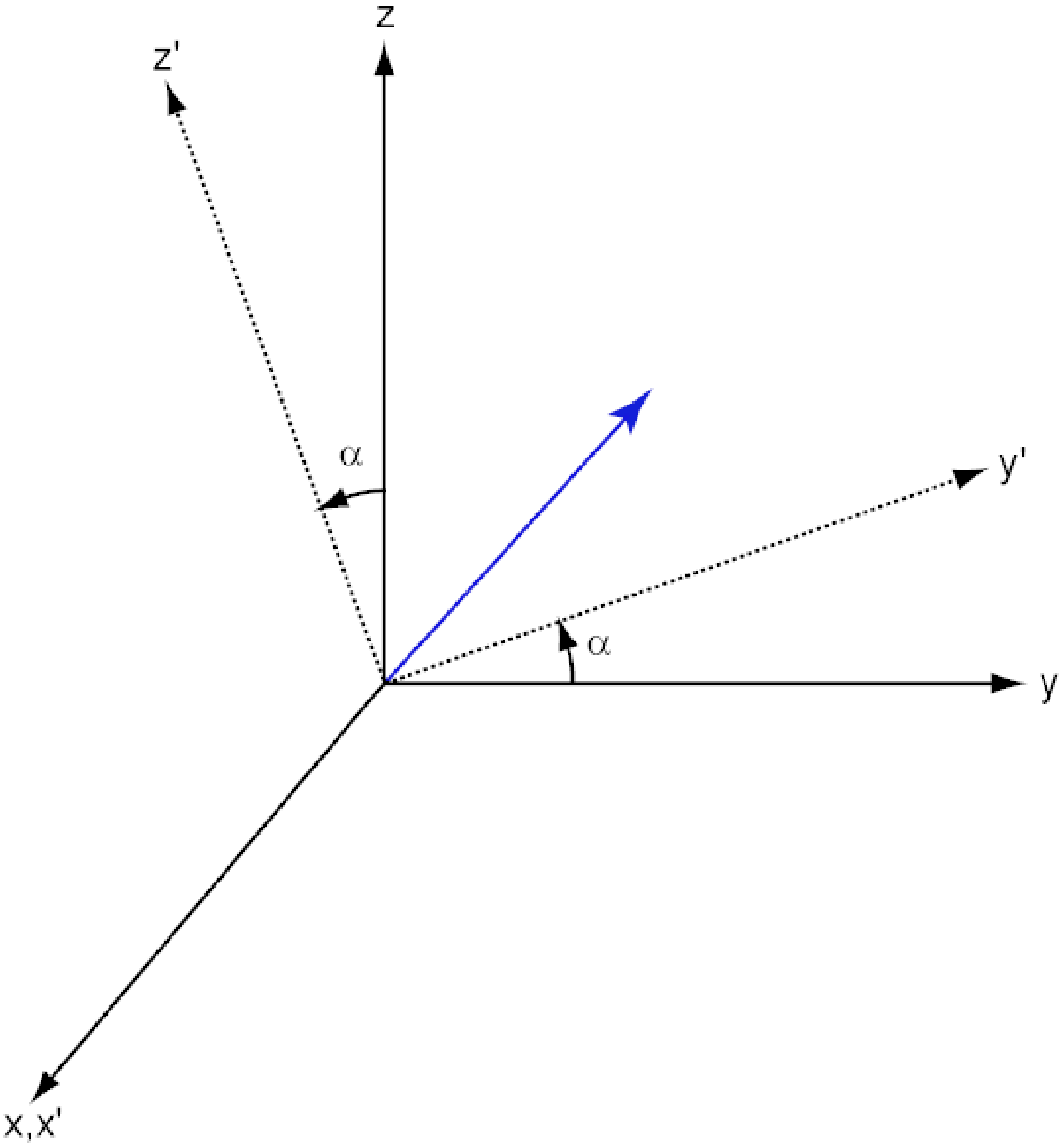
Now any vector can be written as a linear combination of either set of basis vectors:

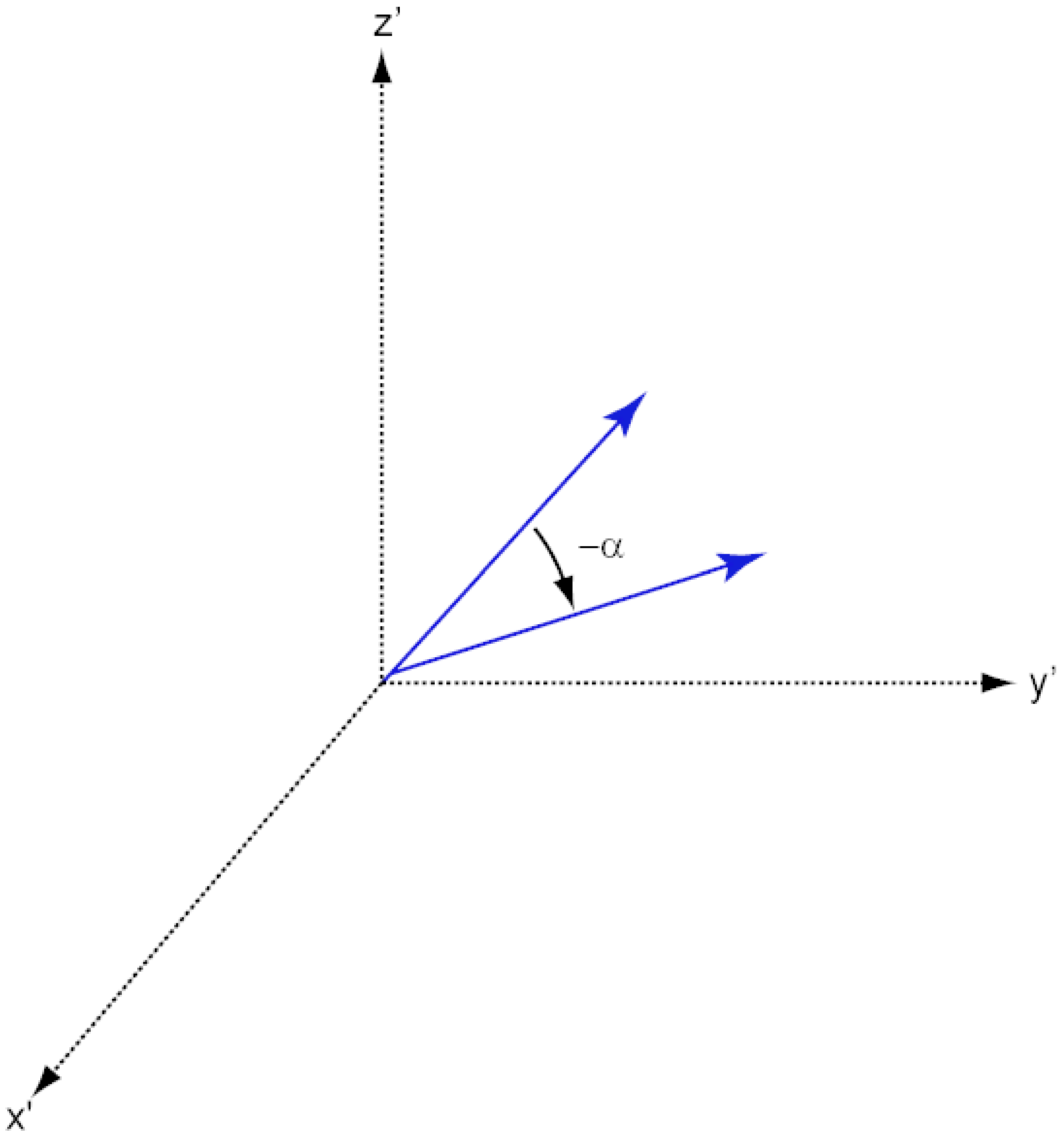
$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k} = v'_x\mathbf{i}' + v'_y\mathbf{j}' + v'_z\mathbf{k}'$$

Using some algebraic manipulation, one can derive the transformation of components for a fixed vector when the basis (or coordinate system) rotates

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = A^{-1} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = A' \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Thus the change in components of a vector when the coordinate system rotates involves the transpose of the rotation matrix. The next figure illustrates how a vector stays fixed as the coordinate system rotates around the x-axis. The figure after shows how this can be interpreted as a rotation *of the vector* in the opposite direction.


z
z'
α
y'
α
y
x,x'


z'
−α
y'
x'

References

[1] Goldstein, H., C. Poole and J. Safko,*Classical Mechanics*, 3rd Edition, San Francisco: Addison Wesley, 2002, pp. 142–144.

See Also

`roty` | `rotz`

Purpose

Rotation matrix for rotations around y-axis

Syntax

`R = roty(ang)`

Description

`R = roty(ang)` creates a 3-by-3 matrix used to rotated a 3-by-1 vector or 3-by-N matrix of vectors around the y-axis by `ang` degrees. When acting on a matrix, each column of the matrix represents a different vector. For the rotation matrix `R` and vector `v`, the rotated vector is given by `R*v`.

Input Arguments

ang - Rotation angle

Real-valued scalar

Rotation angle specified as a real-valued scalar. The rotation angle is positive if the rotation is in the counter-clockwise direction when viewed by an observer looking along the y-axis towards the origin. Angle units are in degrees.

Example: 30.0

Data Types

`double`

Output Arguments

R - Rotation matrix

Real-valued orthogonal matrix

3-by-3 rotation matrix returned as

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

for a rotation angle β.

Examples

Rotation matrix for 45° rotation

Construct the matrix for a rotation of a vector around the y-axis by 45°. Then let the matrix operate on a vector:

```
R = roty(45)

R =

    0.7071         0    0.7071
         0    1.0000         0
   -0.7071         0    0.7071

v = [1;-2;4];
y = R*v

y =

    3.5355
   -2.0000
    2.1213
```

Under a rotation around the y-axis, the y-component of a vector is left unchanged.

Definitions

Rotation Matrices

In transforming vectors in three-dimensional space, rotation matrices are often encountered. Rotation matrices are used in two senses: they can be used to rotate a vector into a new position or they can be used to rotate a coordinate basis (or coordinate system) into a new one. In this case, the vector is left alone but its components in the new basis will be different from those in the original basis. In Euclidean space, there are three basic rotations: one each around the x, y and z axes. Each rotation is specified by an angle of rotation. The rotation angle is defined to be positive for a rotation that is counterclockwise when viewed by an observer looking along the rotation axis towards the origin. Any arbitrary rotation can be composed of a combination of these three *(Euler's rotation theorem)*. For example, one can rotated a vector using a sequence of three rotations: $\mathbf{v}' = A\mathbf{v} = R_z(\gamma)R_y(\beta)R_x(\alpha)\mathbf{v}$.

The rotation matrices that rotate a vector around the x, y, and z-axes are given by:

- Counterclockwise rotation around x-axis

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

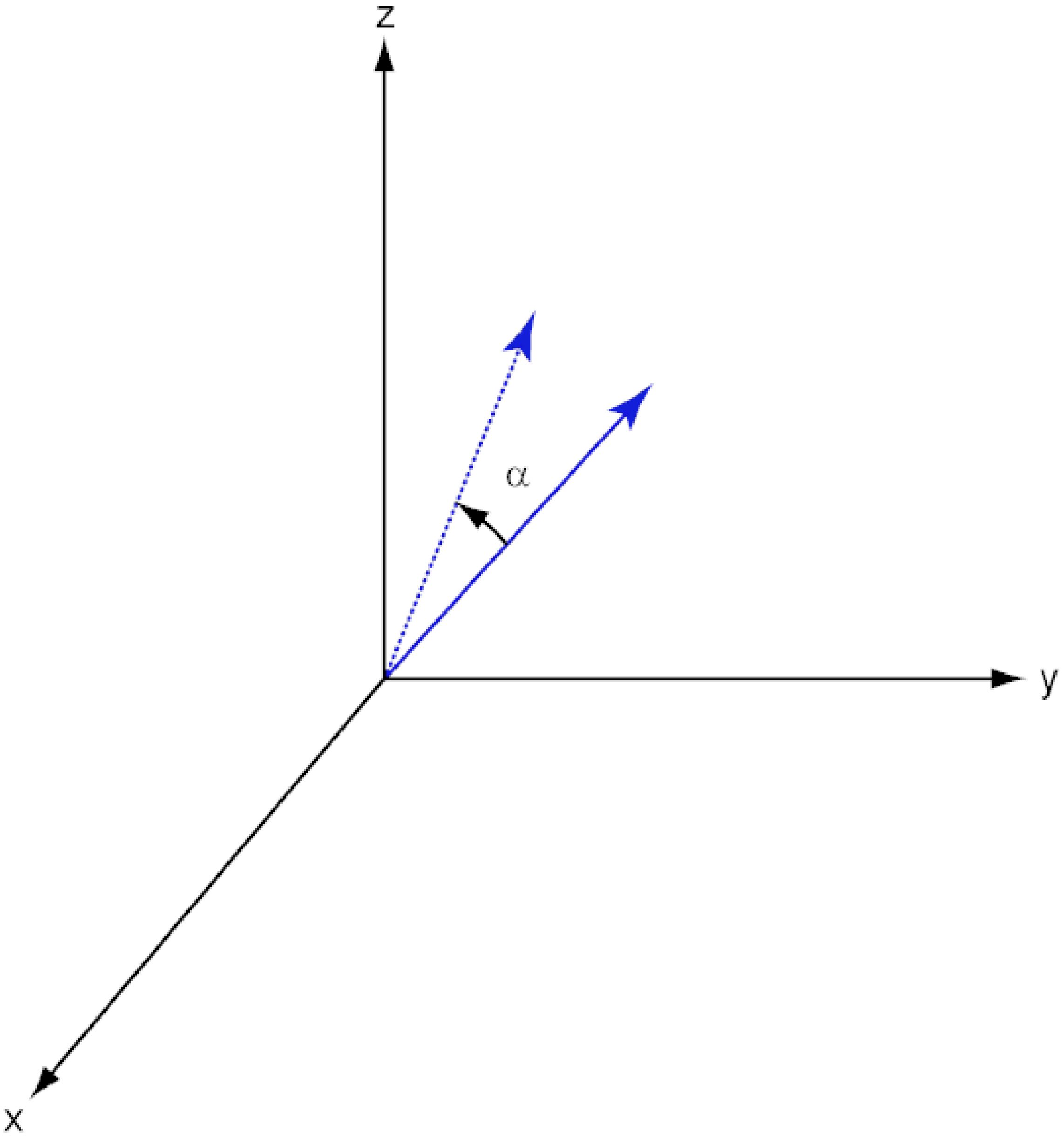
- Counterclockwise rotation around y-axis

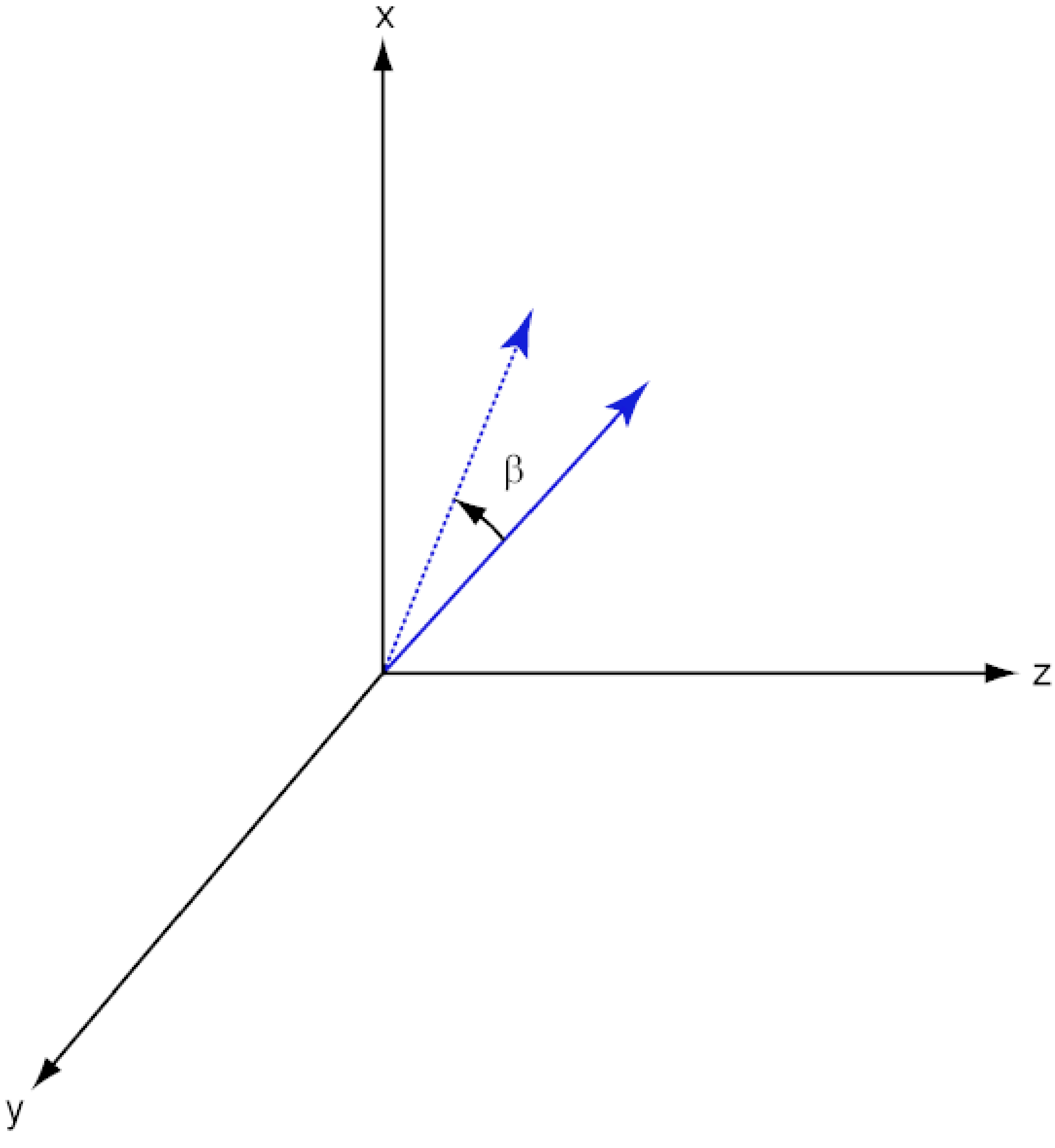
$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

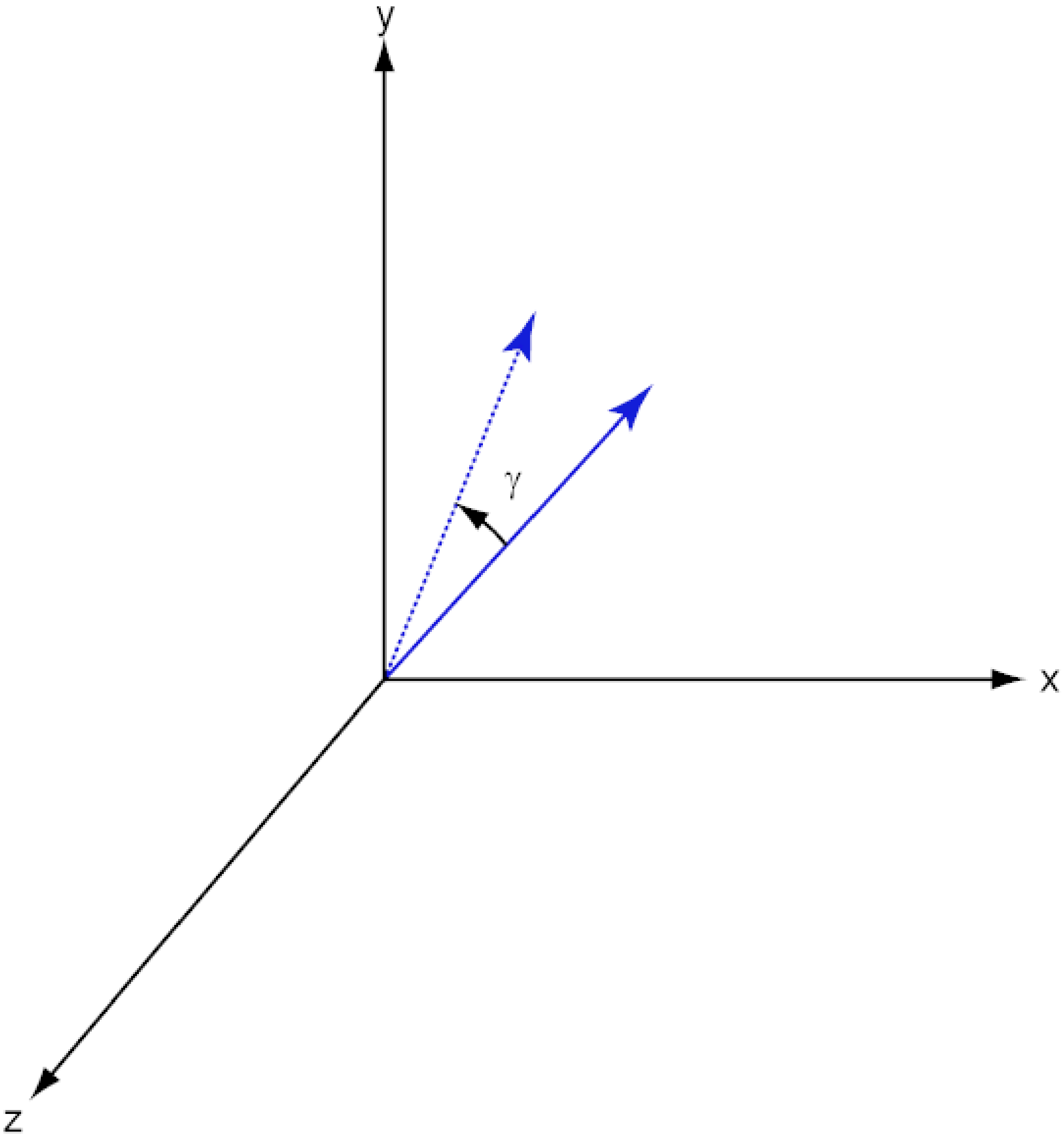
- Counterclockwise rotation around z-axis

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The following three figures show what positive rotations look like for each rotation axis:


z
y
x
α


x
β
z
y


y
x
z
γ

For any rotation, there is an inverse rotation satisfying $A^{-1}A = 1$. For example, the inverse of the x-axis rotation matrix is obtained by changing the sign of the angle:

$$R_x^{-1}(\alpha) = R_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} = R_x^{'}(\alpha)$$

This example illustrates a basic property: the inverse rotation matrix equals the transpose of the original. Rotation matrices satisfy *A'A = 1*, and consequently *det(A) = 1.* Under rotations, vector lengths are preserved as well as the angles between vectors.

We can think of rotations in another way. Consider the original set of basis vectors, $\mathbf{i}, \mathbf{j}, \mathbf{k}$, and rotate them all using the rotation matrix *A*.

This produces a new set of basis vectors $\mathbf{i}^{'}, \mathbf{j}, ^{'}\mathbf{k}^{'}$ related to the original by:

$$\begin{aligned} \mathbf{i}' &= A\mathbf{i} \\ \mathbf{j}' &= A\mathbf{j} \\ \mathbf{k}' &= A\mathbf{k} \end{aligned}$$

The new basis vectors can be written as linear combinations of the old ones and involve the transpose:

$$\begin{bmatrix} \mathbf{i}' \\ \mathbf{j}' \\ \mathbf{k}' \end{bmatrix} = A' \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}$$

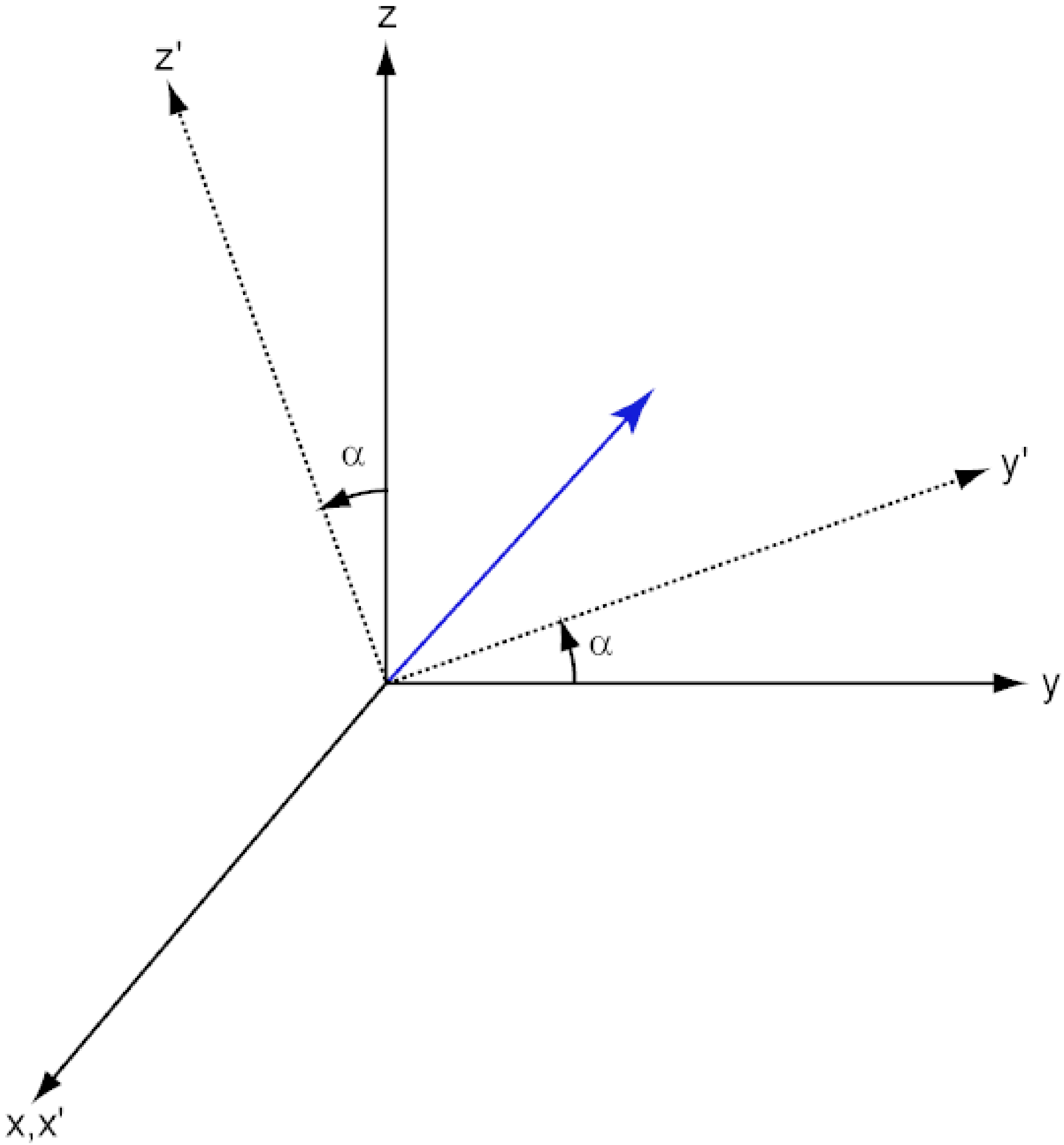
Now any vector can be written as a linear combination of either set of basis vectors:

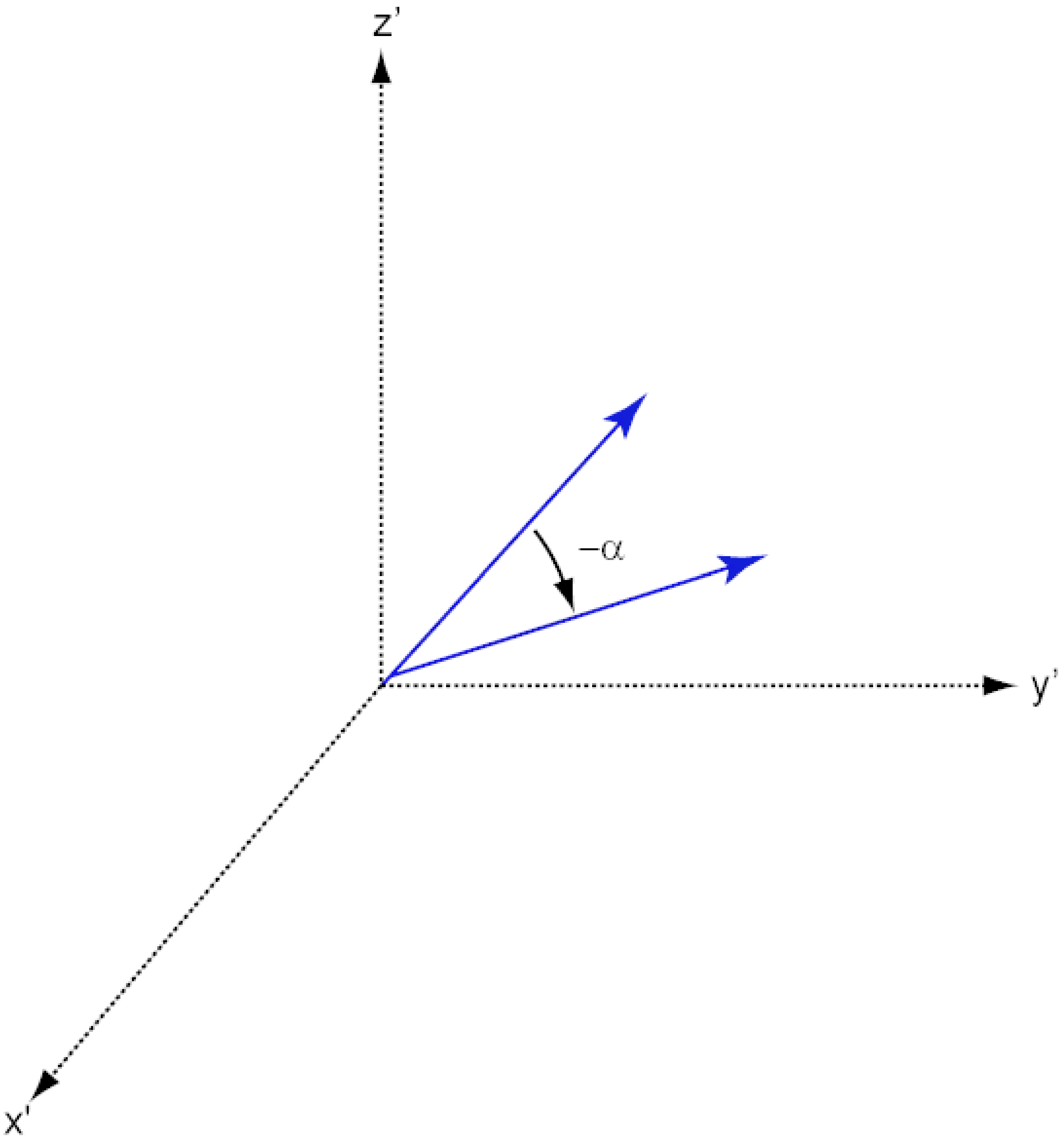
$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k} = v'_x\mathbf{i}' + v'_y\mathbf{j}' + v'_z\mathbf{k}'$$

Using some algebraic manipulation, one can derive the transformation of components for a fixed vector when the basis (or coordinate system) rotates

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = A^{-1} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = A' \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Thus the change in components of a vector when the coordinate system rotates involves the transpose of the rotation matrix. The next figure illustrates how a vector stays fixed as the coordinate system rotates around the x-axis. The figure after shows how this can be interpreted as a rotation *of the vector* in the opposite direction.


z
z'
α
y'
α
y
x,x'


z'
y'
x'
−α

References

[1] Goldstein, H., C. Poole and J. Safko, *Classical Mechanics*, 3rd Edition, San Francisco: Addison Wesley, 2002, pp. 142–144.

See Also

`rotx` | `rotz`

Purpose

Rotation matrix for rotations around z-axis

Syntax

`R = rotz(ang)`

Description

`R = rotz(ang)` creates a 3-by-3 matrix used to rotated a 3-by-1 vector or 3-by-N matrix of vectors around the z-axis by `ang` degrees. When acting on a matrix, each column of the matrix represents a different vector. For the rotation matrix `R` and vector `v`, the rotated vector is given by `R*v`.

Input Arguments

ang - Rotation angle

Real-valued scalar

Rotation angle specified as a real-valued scalar. The rotation angle is positive if the rotation is in the counter-clockwise direction when viewed by an observer looking along the z-axis towards the origin. Angle units are in degrees.

Example: 45.0

Data Types
`double`

Output Arguments

R - Rotation matrix

Real-valued orthogonal matrix

3-by-3 rotation matrix returned as

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

for a rotation angle γ.

Examples

Rotation matrix for 45° rotation

Construct the matrix for a rotation of a vector around the z-axis by 45°. Then let the matrix operate on a vector:

```
R = rotz(45)

R =

    0.7071   -0.7071         0
    0.7071    0.7071         0
         0         0    1.0000

v = [1;-2;4];
y = R*v

y =

    2.1213
   -0.7071
    4.0000
```

Under a rotation around the z-axis, the z-component of a vector is left unchanged.

Definitions

Rotation Matrices

In transforming vectors in three-dimensional space, rotation matrices are often encountered. Rotation matrices are used in two senses: they can be used to rotate a vector into a new position or they can be used to rotate a coordinate basis (or coordinate system) into a new one. In this case, the vector is left alone but its components in the new basis will be different from those in the original basis. In Euclidean space, there are three basic rotations: one each around the x, y and z axes. Each rotation is specified by an angle of rotation. The rotation angle is defined to be positive for a rotation that is counterclockwise when viewed by an observer looking along the rotation axis towards the origin. Any arbitrary rotation can be composed of a combination of these three *(Euler's rotation theorem)*. For example, one can rotated a vector using a sequence of three rotations: $\mathbf{v}' = A\mathbf{v} = R_z(\gamma)R_y(\beta)R_x(\alpha)\mathbf{v}$.

The rotation matrices that rotate a vector around the x, y, and z-axes are given by:

- Counterclockwise rotation around x-axis

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$

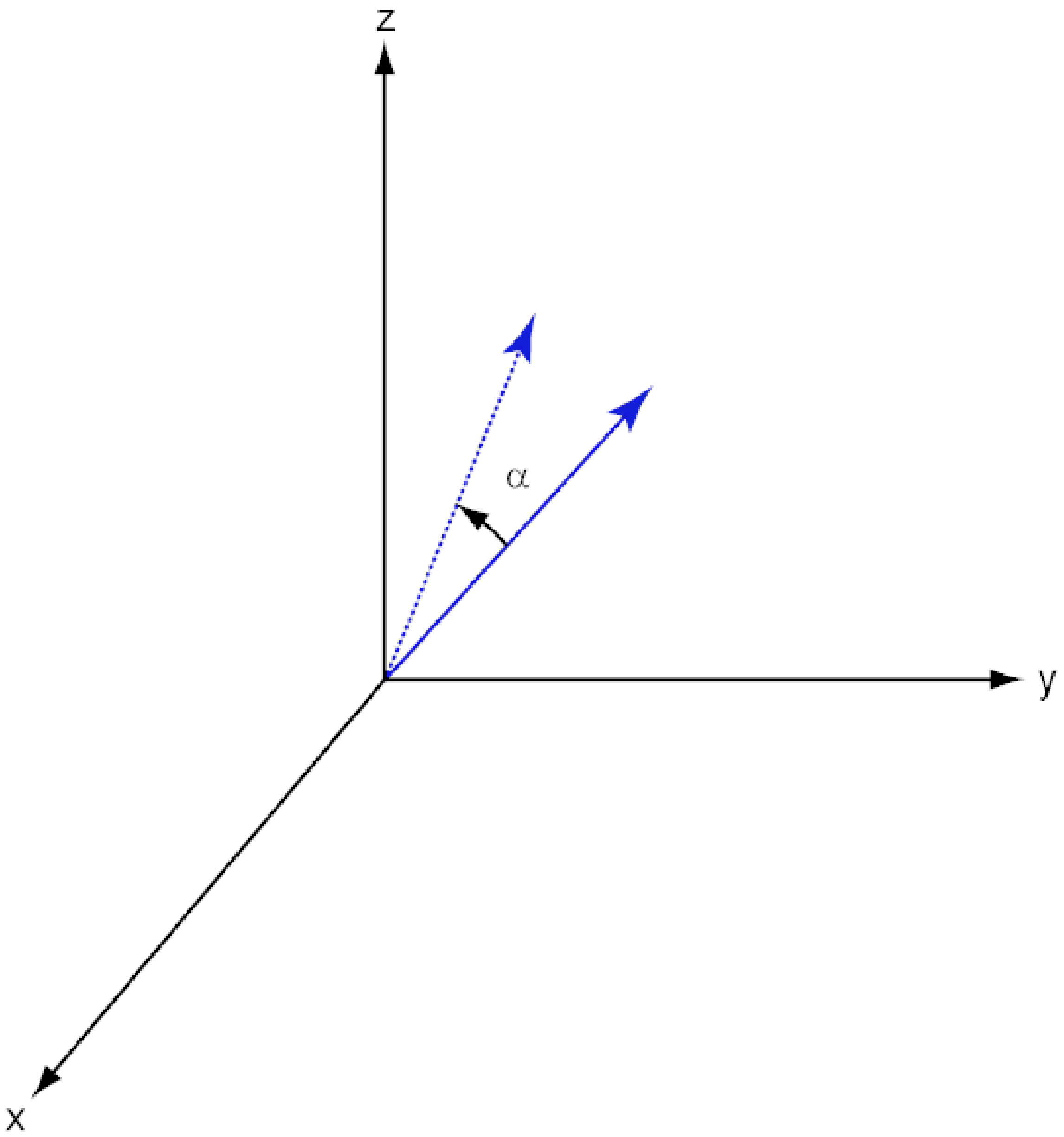
- Counterclockwise rotation around y-axis

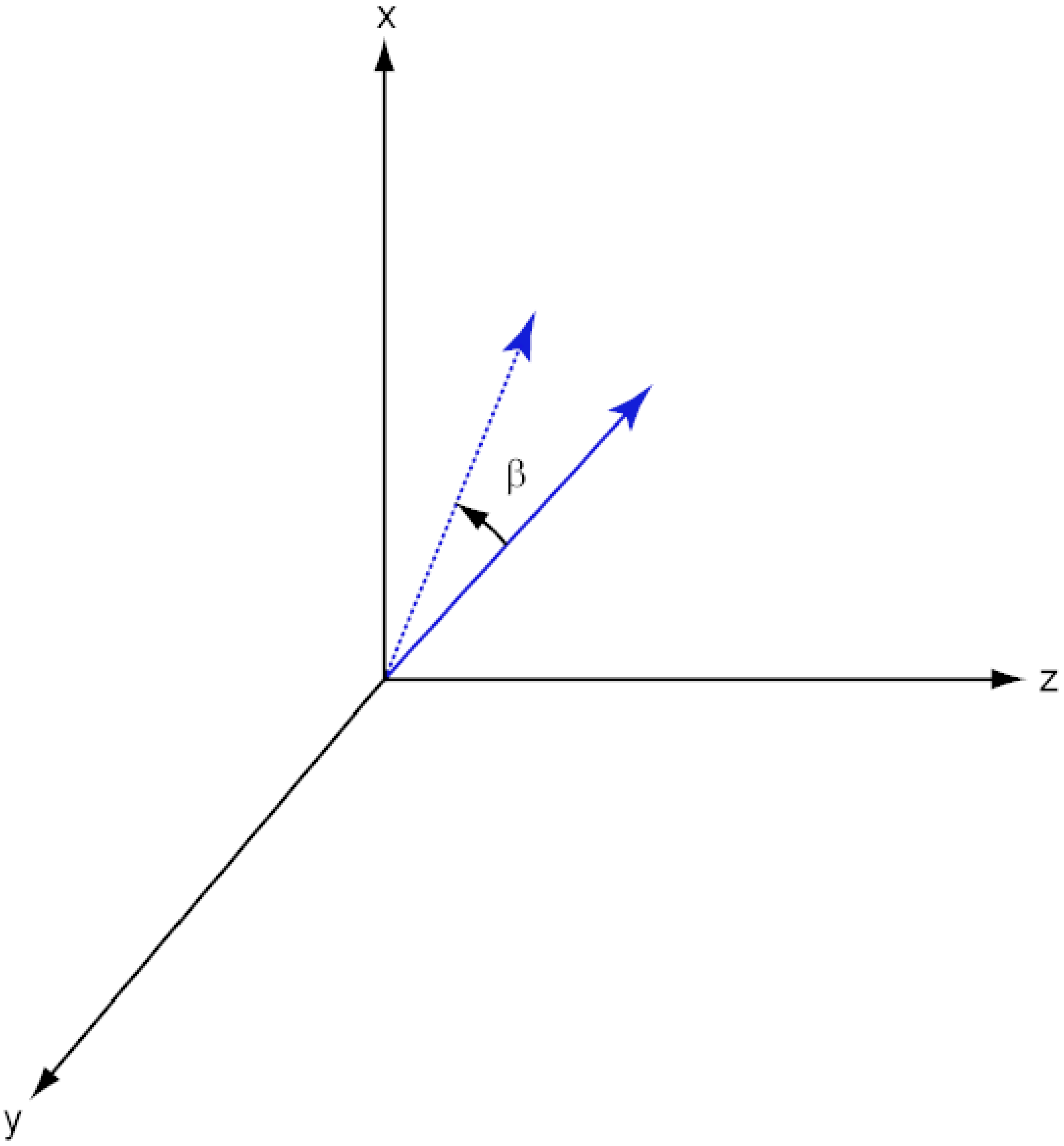
$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}$$

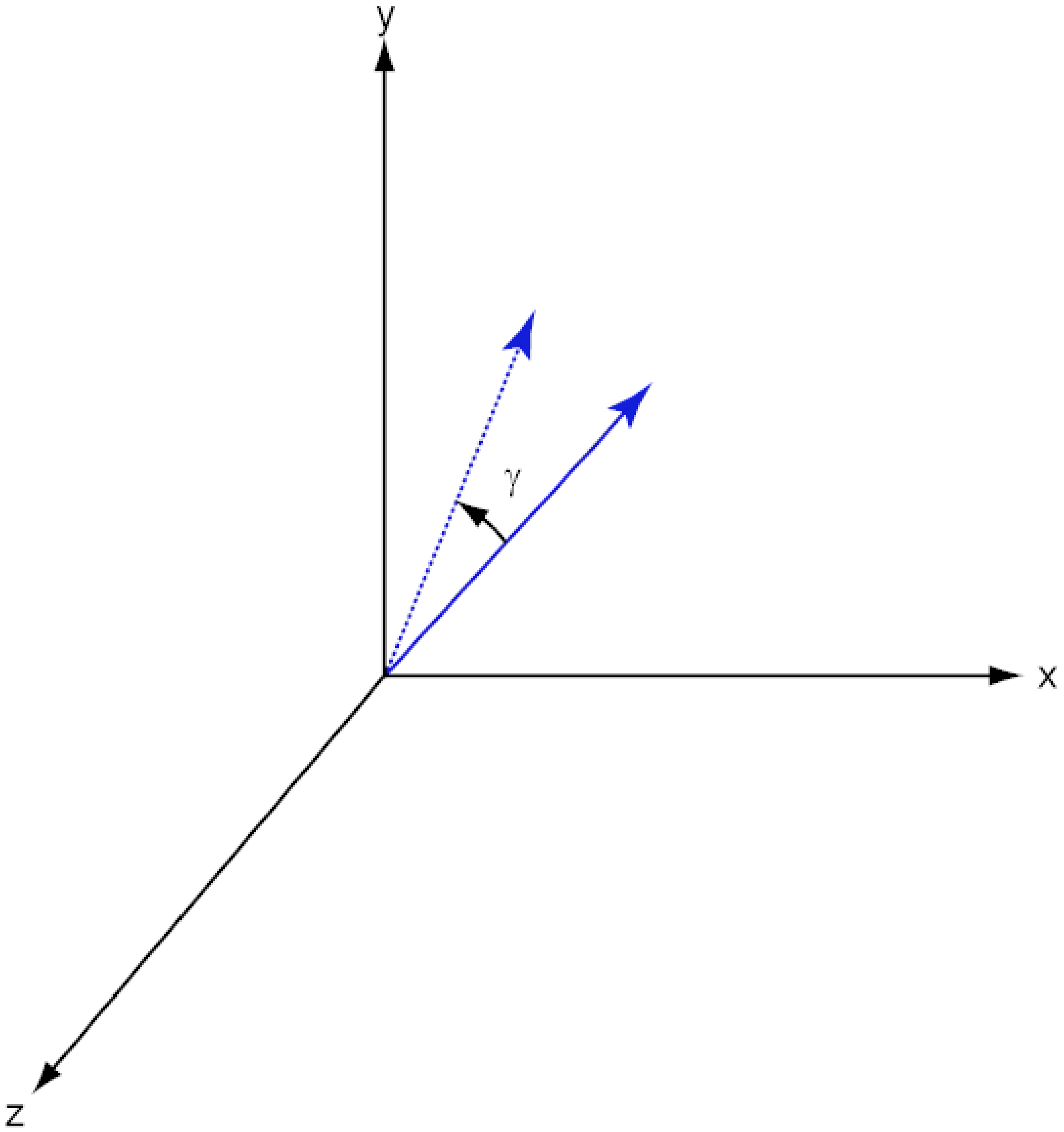
- Counterclockwise rotation around z-axis

$$R_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The following three figures show what positive rotations look like for each rotation axis:


z
y
x
α


x
β
z
y


y
x
z
γ

For any rotation, there is an inverse rotation satisfying $A^{-1}A = 1$. For example, the inverse of the x-axis rotation matrix is obtained by changing the sign of the angle:

$$R_x^{-1}(\alpha) = R_x(-\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} = R_x'(\alpha)$$

This example illustrates a basic property: the inverse rotation matrix equals the transpose of the original. Rotation matrices satisfy *A'A = 1*, and consequently *det(A) = 1.* Under rotations, vector lengths are preserved as well as the angles between vectors.

We can think of rotations in another way. Consider the original set of basis vectors, $\mathbf{i}, \mathbf{j}, \mathbf{k}$, and rotate them all using the rotation matrix *A*.

This produces a new set of basis vectors $\mathbf{i}', \mathbf{j}', \mathbf{k}'$ related to the original by:

$$\begin{aligned} \mathbf{i}' &= A\mathbf{i} \\ \mathbf{j}' &= A\mathbf{j} \\ \mathbf{k}' &= A\mathbf{k} \end{aligned}$$

The new basis vectors can be written as linear combinations of the old ones and involve the transpose:

$$\begin{bmatrix} \mathbf{i}' \\ \mathbf{j}' \\ \mathbf{k}' \end{bmatrix} = A' \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}$$

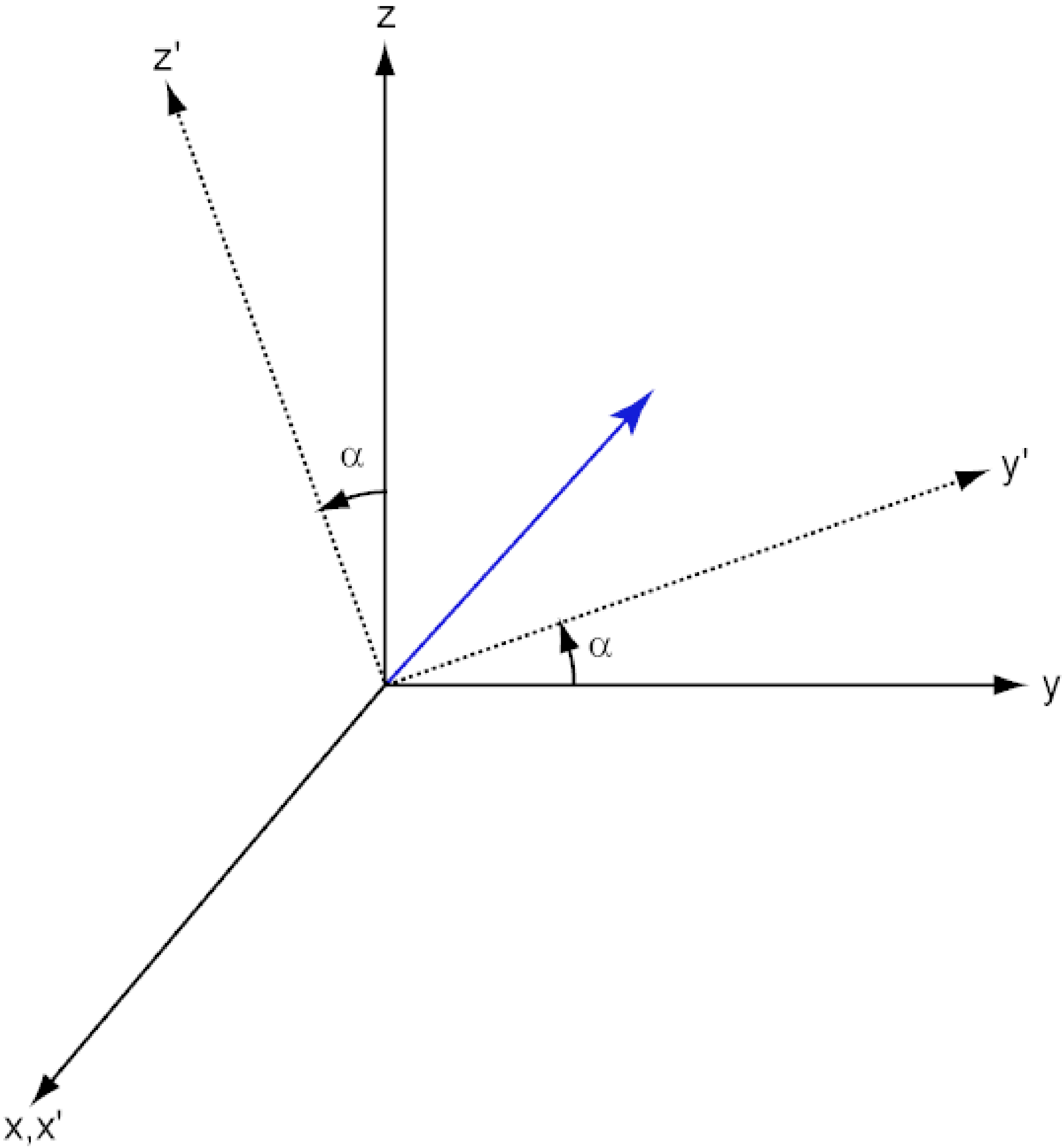
Now any vector can be written as a linear combination of either set of basis vectors:

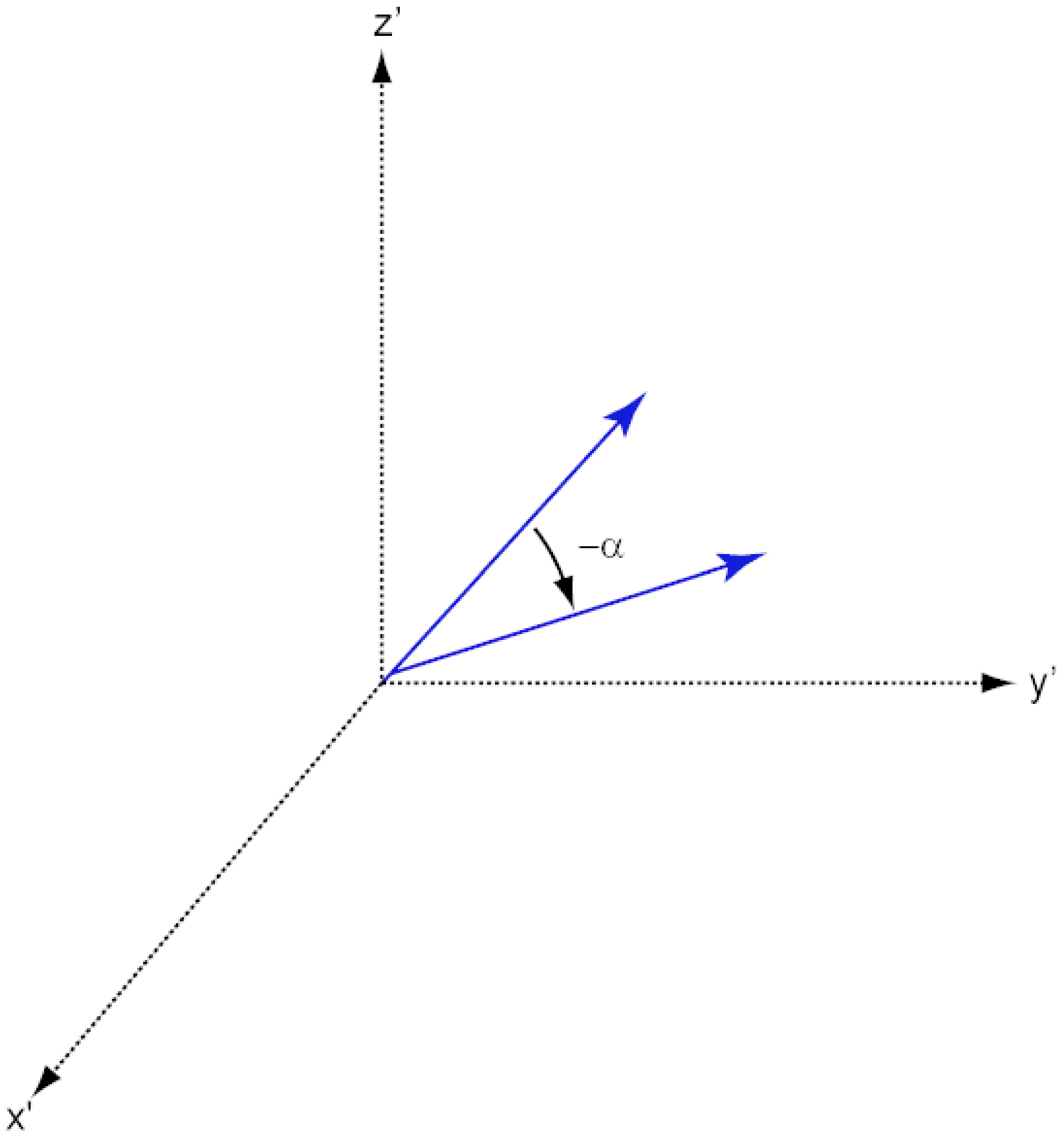
$$\mathbf{v} = v_x\mathbf{i} + v_y\mathbf{j} + v_z\mathbf{k} = v'_x\mathbf{i}' + v'_y\mathbf{j}' + v'_z\mathbf{k}'$$

Using some algebraic manipulation, one can derive the transformation of components for a fixed vector when the basis (or coordinate system) rotates

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \end{bmatrix} = A^{-1} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = A' \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

Thus the change in components of a vector when the coordinate system rotates involves the transpose of the rotation matrix. The next figure illustrates how a vector stays fixed as the coordinate system rotates around the x-axis. The figure after shows how this can be interpreted as a rotation *of the vector* in the opposite direction.


z
z'
α
y'
α
y
x,x'


z'
−α
y'
x'

References

[1] Goldstein, H., C. Poole and J. Safko, *Classical Mechanics*, 3rd Edition, San Francisco: Addison Wesley, 2002, pp. 142–144.

See Also

`rotx` | `roty`

sensorArrayAnalyzer

Purpose

Sensor array analyzer

Description

You use the **Sensor Array Analyzer** app to construct and analyze common sensor array configurations. These configurations range from 1-D to 3-D arrays of antennas and microphones. You can use this app to generate the directivity of the following arrays.

Uniform Linear Array (ULA)
Uniform Circular Array

Circular Plane Array
Spherical Array
Arbitrary Geometry

Uniform Rectangular Array (URA)
Uniform Hexagonal Array

Concentric Array
Cylindrical Array

Each array type has different sets of parameters for its specification. After you select an array type, you can modify the array parameters. The parameters you can set include the type of antenna or microphone elements, the number and spacing of elements, and any array *tapering* (also called *shading*). You can enter the element spacing in meters or units of wavelength. After you enter all the information for your array, the app then displays basic performance characteristics, such as array directivity and array dimensions.

These are the types of elements available to populate an array.

Isotropic Antenna
Cosine Antenna
Omnidirectional Microphone
Cardioid Microphone
Custom Antenna

The **Sensor Array Analyzer** app lets you produce a variety of plots and images. These are types of plots available.

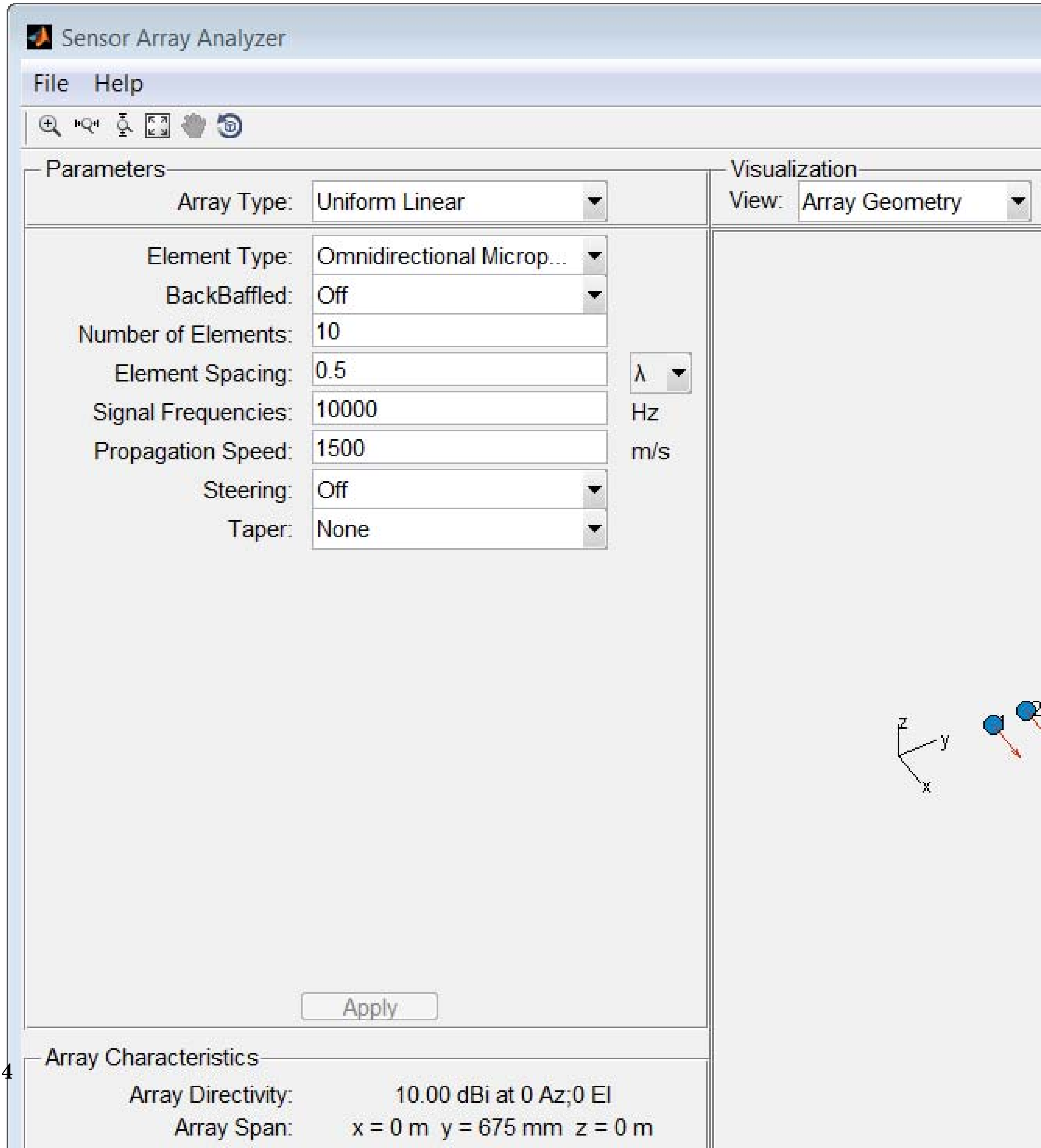
Plot type	
Array Geometry	
2D Array Directivity	
3D Array Directivity	
Grating Lobes	Available for the Uniform Linear Array, the Uniform Rectangular Array, the Uniform Hexagonal Array, and the Circular Planar Array.

Examples

Uniform Linear Array

Start with 10-element uniform linear array (ULA) in a sonar application with omnidirectional microphones. A uniform linear array has its sensor elements equally-spaced spaced along a single line. Set the **Array Type** to `Uniform Linear` and the **Element Type** to `Omnidirectional Microphone`. Design the array to find the arrival direction of a 10 kHz signal by setting **Signal Frequencies** to `10000` and the **Element Spacing** to 0.5 wavelengths. In water, for example, you can set the signal **Propagation Speed** to equal the speed of sound in water, `1500` m/s.

Then, in the **View** dropdown menu, choose the `Array Geometry` option to draw the shape of the array.



Sensor Array Analyzer
File
Help
Apply
z
y
x
1
2

Parameters

Array Type: Uniform Linear

Element Type: Omnidirectional Microp...

BackBaffled: Off

Number of Elements: 10

Element Spacing: 0.5 λ

Signal Frequencies: 10000 Hz

Propagation Speed: 1500 m/s

Steering: Off

Taper: None

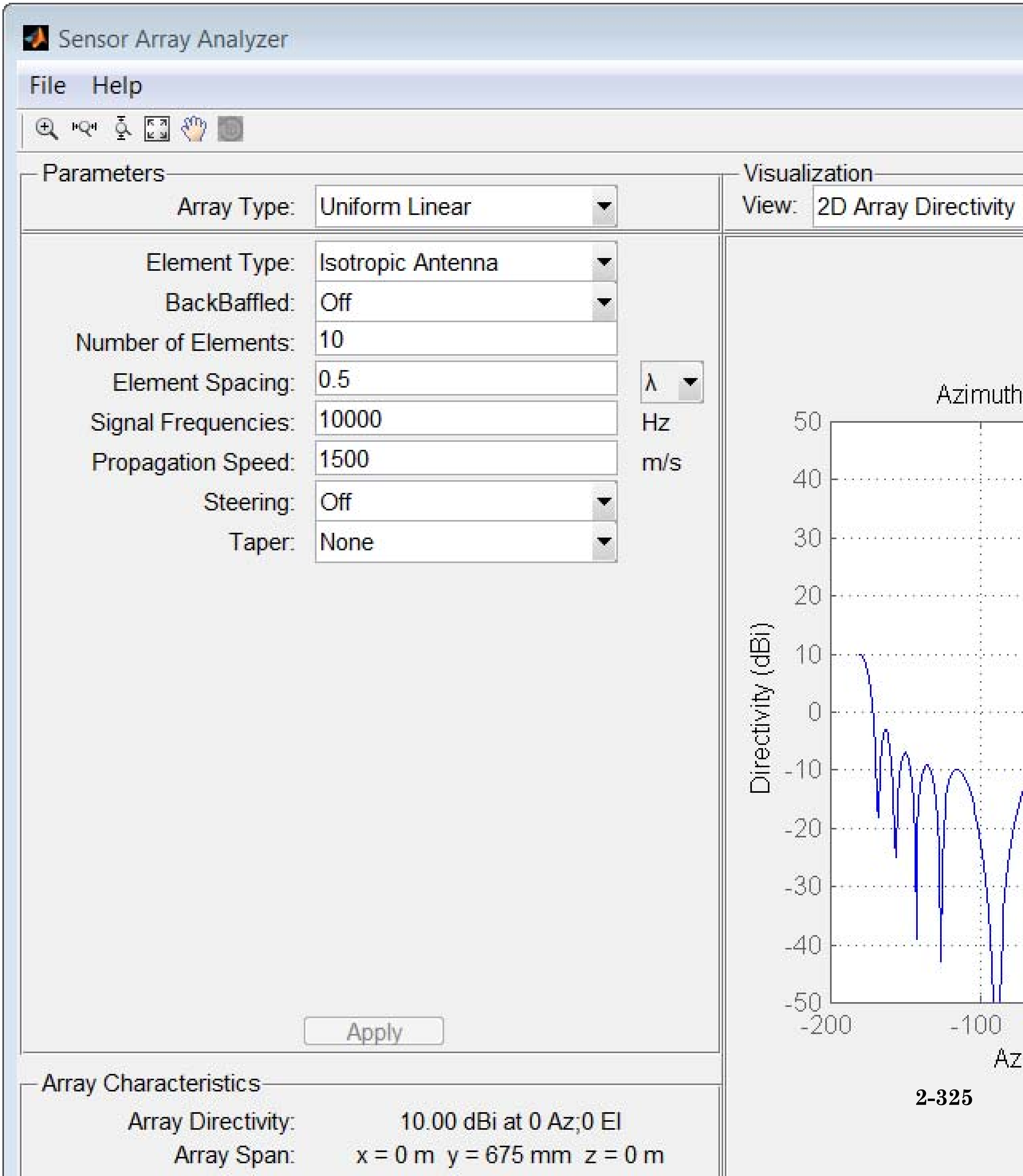
Visualization

View: Array Geometry

Array Characteristics

Array Directivity: 10.00 dBi at 0 Az;0 El

Array Span: x = 0 m y = 675 mm z = 0 m


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Azimuth
Directivity (dBi)
50
40
30
20
10
0
-10
-20
-30
-40
-50
-200
-100
Az

Array Type: Uniform Linear

Element Type: Isotropic Antenna

BackBaffled: Off

Number of Elements: 10

Element Spacing: 0.5 λ

Signal Frequencies: 10000 Hz

Propagation Speed: 1500 m/s

Steering: Off

Taper: None

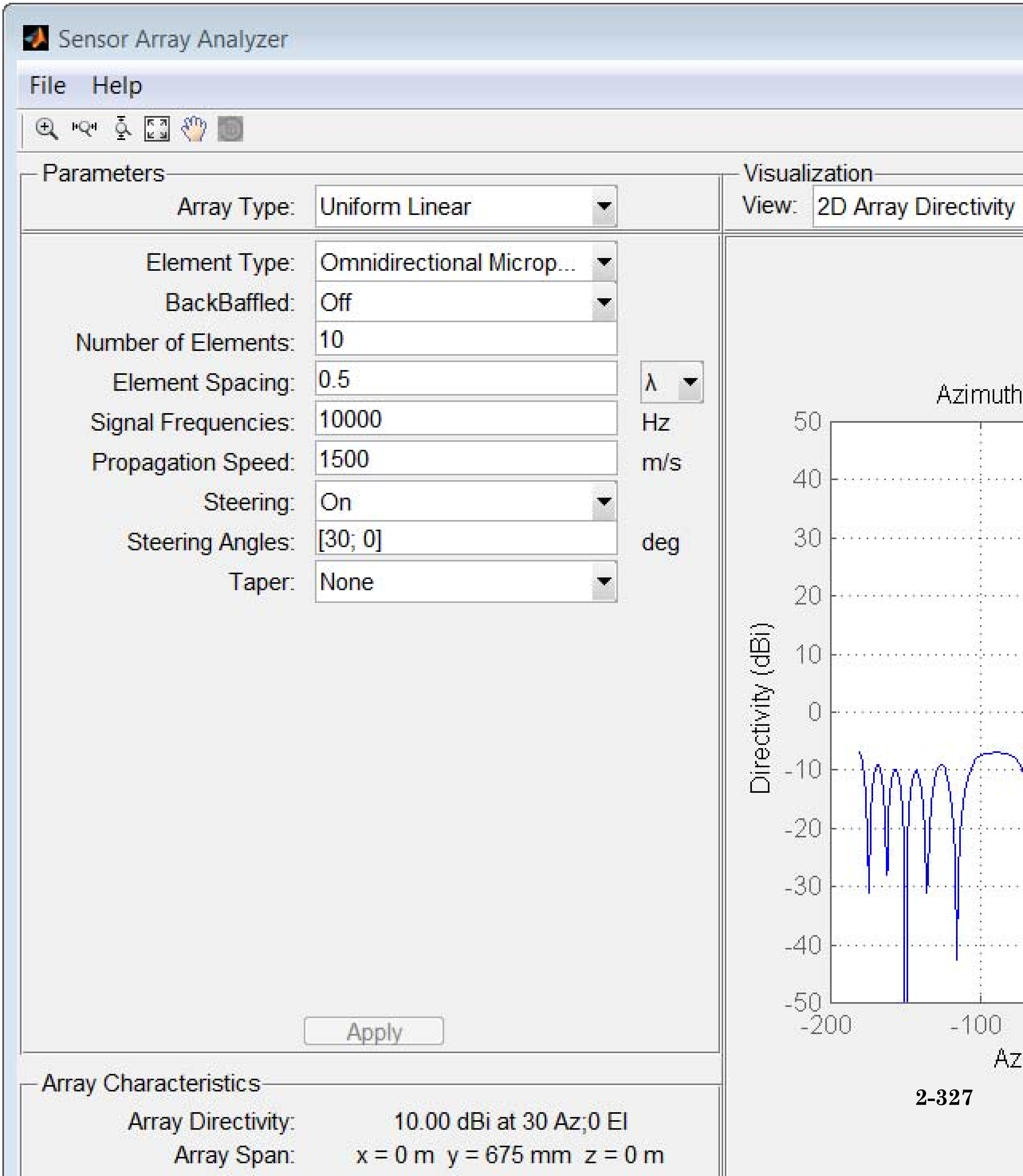
View: 2D Array Directivity

Array Characteristics

Array Directivity: 10.00 dBi at 0 Az;0 El

Array Span: x = 0 m y = 675 mm z = 0 m

A beamscanner works by successively pointing the array mainlobe in a sequence of different directions. Setting the **Steering** option to `On` lets you steer the mainlobe in the direction specified by the **Steering Angles** option. In this case, set the steering angle to `[30;0]` to point the mainlobe to 30° in azimuth and 0° elevation. In the next figure, you can see two mainlobes, one at 30° as expected, and another at 150°. Again, two mainlobes appear because of the cylindrical symmetry of the array.


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Azimuth
Directivity (dBi)
50
40
30
20
10
0
-10
-20
-30
-40
-50
-200
-100
Az

Array Type: Uniform Linear

Element Type: Omnidirectional Microp...

BackBaffled: Off

Number of Elements: 10

Element Spacing: 0.5 λ

Signal Frequencies: 10000 Hz

Propagation Speed: 1500 m/s

Steering: On

Steering Angles: [30; 0] deg

Taper: None

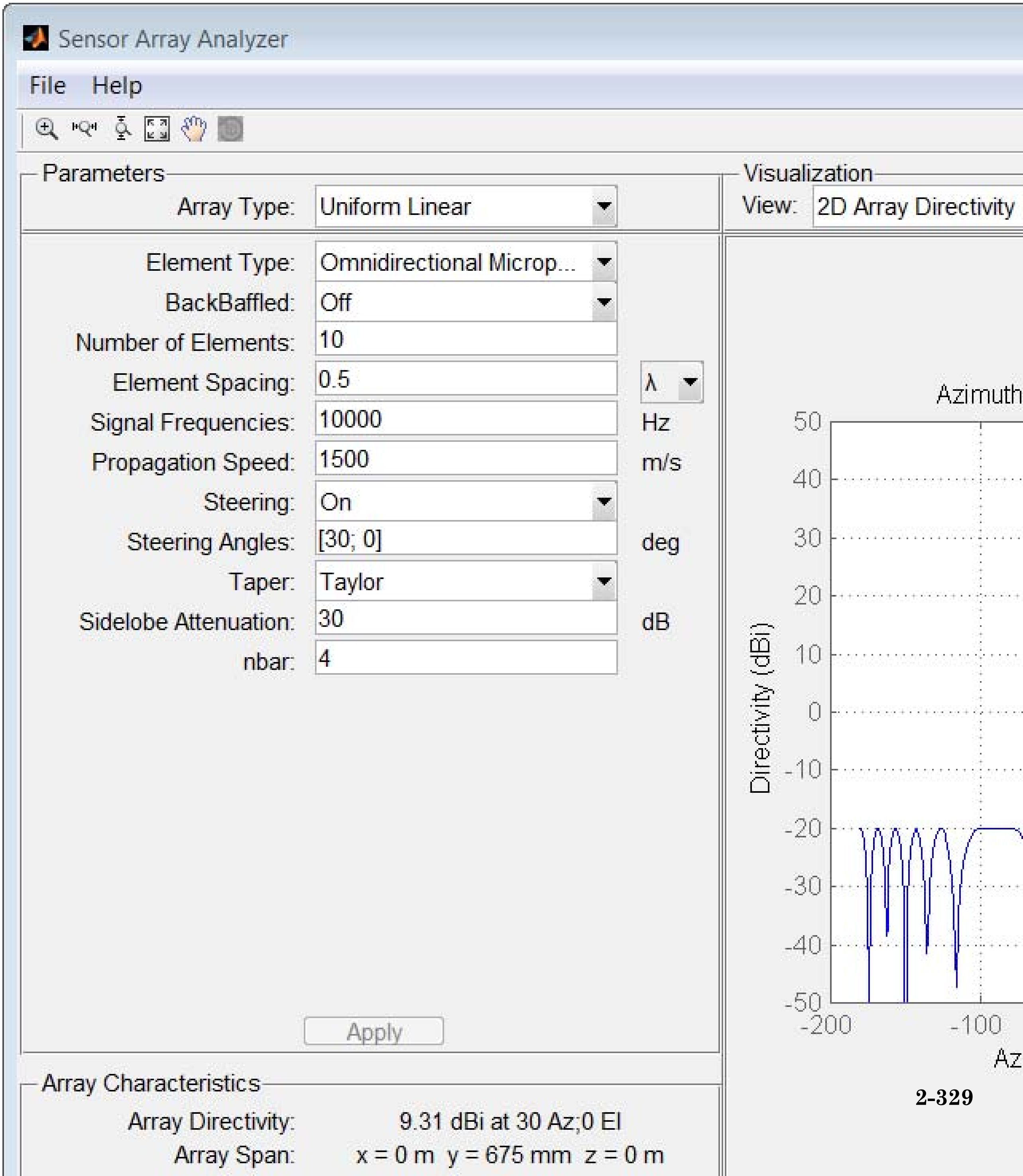
View: 2D Array Directivity

Array Characteristics

Array Directivity: 10.00 dBi at 30 Az;0 El

Array Span: x = 0 m y = 675 mm z = 0 m

down by about only 13 dB. A strong sidelobe inhibits the ability of the array to detect a weaker signal in the presence of a larger nearby signal. By using array tapering, you can reduce the side lobes. Use the **Taper** option to specify the array taper as a `Taylor` window with **Sidelobe Attenuation** set to `30` dB. The next figure shows how the Taylor window reduces all side lobes to –30 dB—but at the expense of broadening the mainlobe.


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Azimuth
50
40
30
20
10
0
-10
-20
-30
-40
-50
-200
-100
Directivity (dBi)
Az

Array Type: Uniform Linear

Element Type: Omnidirectional Microp...

BackBaffled: Off

Number of Elements: 10

Element Spacing: 0.5 λ

Signal Frequencies: 10000 Hz

Propagation Speed: 1500 m/s

Steering: On

Steering Angles: [30; 0] deg

Taper: Taylor

Sidelobe Attenuation: 30 dB

nbar: 4

View: 2D Array Directivity

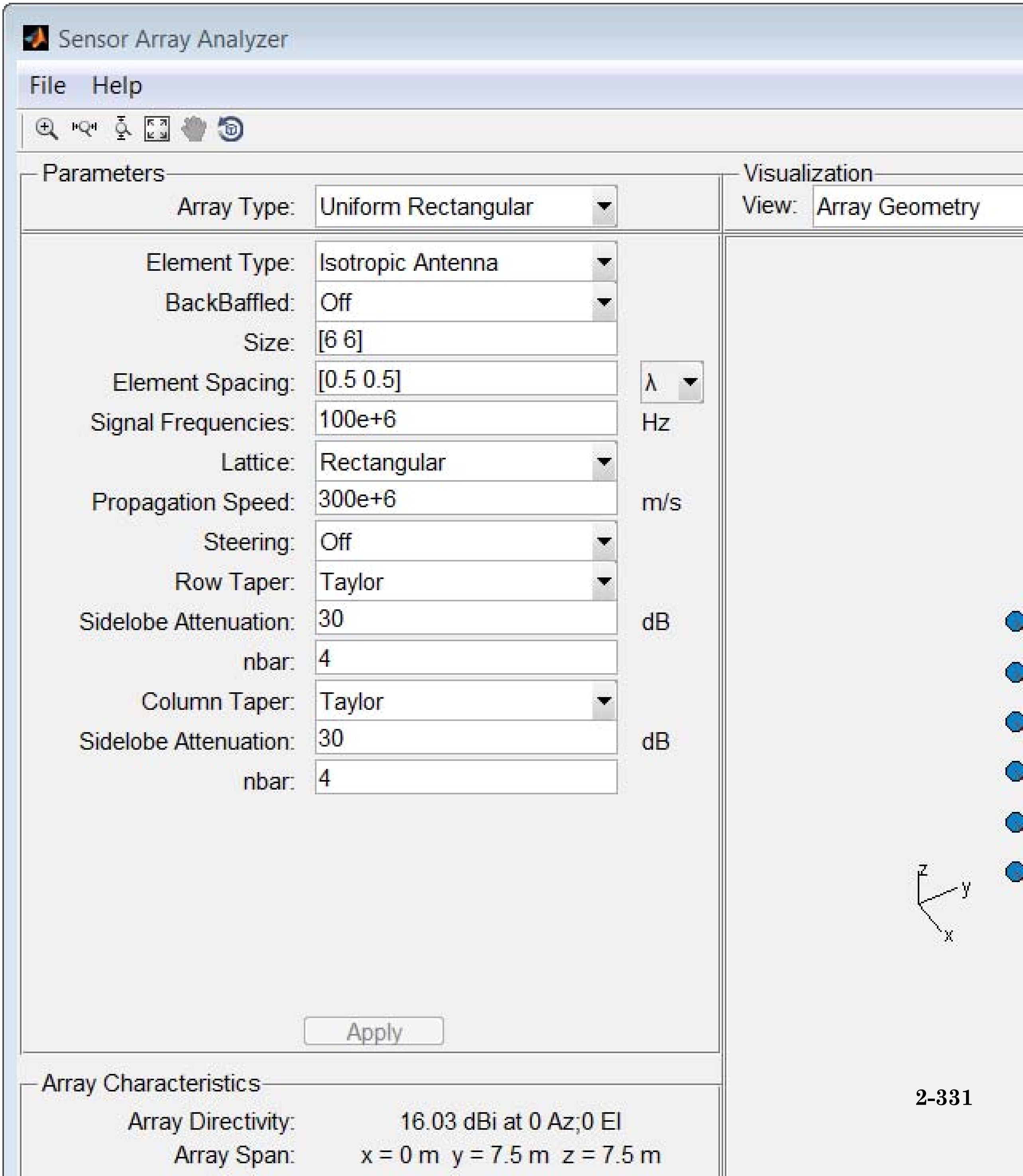
Array Characteristics

Array Directivity: 9.31 dBi at 30 Az;0 El

Array Span: x = 0 m y = 675 mm z = 0 m

Uniform Rectangular Array

Construct a 6-by-6 uniform rectangular array (URA) designed to detect and localize a 100 MHz signal. Set the **Array Type** to `Uniform Rectangular`, the **Element Type** to `Isotropic Antenna`, and the **Size** to `[6 6]`. Design the array to find the arrival direction of a 100 MHz signal by setting **Signal Frequencies** to `100e+6` and the row and column **Element Spacing** to 0.5 wavelength. Set both the **Row Taper** and **Column Taper** to a `Taylor` window. The shape of the array is shown in the figure below.


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
z
y
x

Array Type: Uniform Rectangular

View: Array Geometry

Element Type: Isotropic Antenna

BackBaffled: Off

Size: [6 6]

Element Spacing: [0.5 0.5] λ

Signal Frequencies: 100e+6 Hz

Lattice: Rectangular

Propagation Speed: 300e+6 m/s

Steering: Off

Row Taper: Taylor

Sidelobe Attenuation: 30 dB

nbar: 4

Column Taper: Taylor

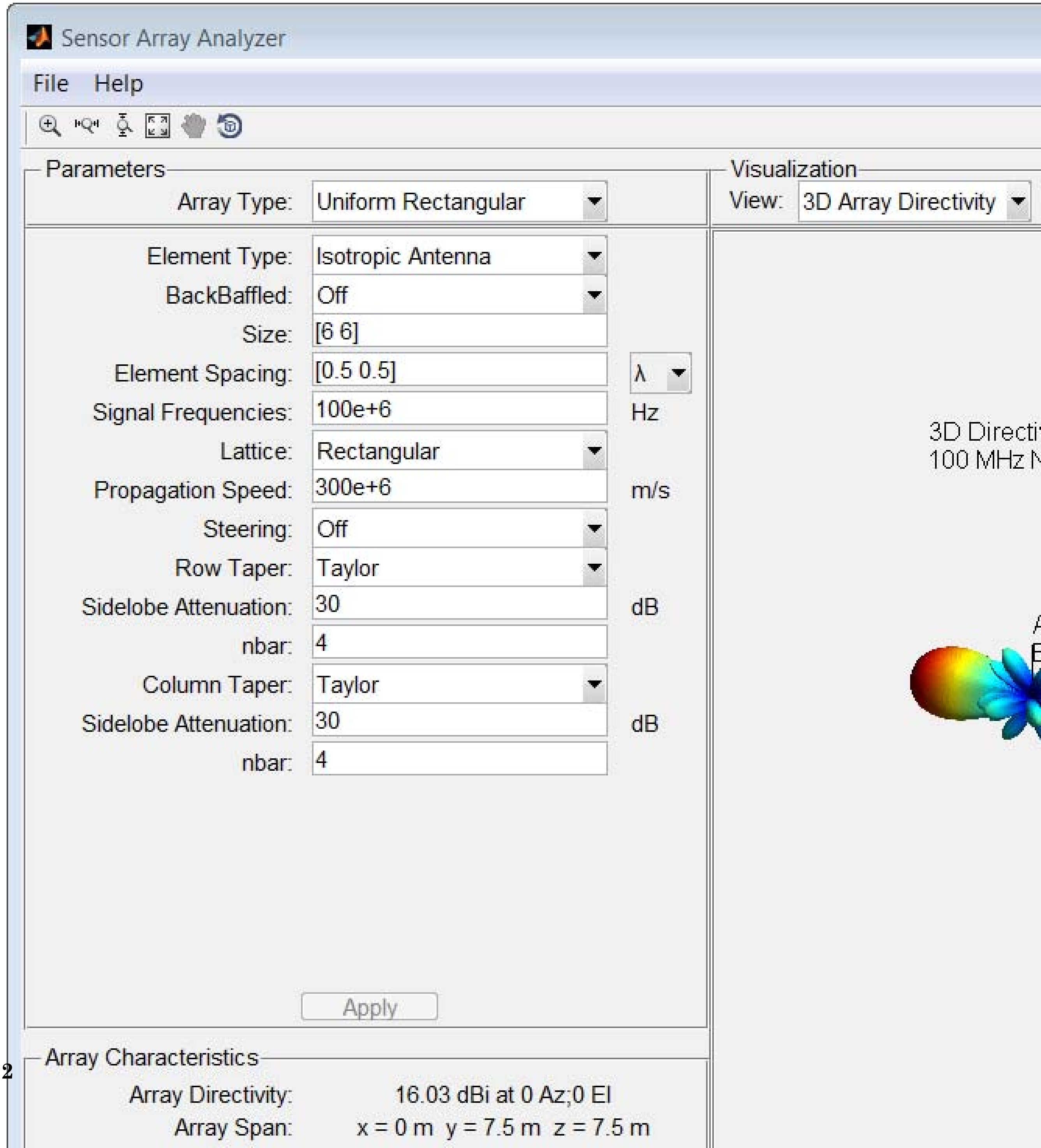
Sidelobe Attenuation: 30 dB

nbar: 4

Array Characteristics

Array Directivity: 16.03 dBi at 0 Az;0 El

Array Span: x = 0 m y = 7.5 m z = 7.5 m


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Array Characteristics

Array Type: Uniform Rectangular

Element Type: Isotropic Antenna
BackBaffled: Off
Size: [6 6]
Element Spacing: [0.5 0.5] λ
Signal Frequencies: 100e+6 Hz
Lattice: Rectangular
Propagation Speed: 300e+6 m/s
Steering: Off
Row Taper: Taylor
Sidelobe Attenuation: 30 dB
nbar: 4
Column Taper: Taylor
Sidelobe Attenuation: 30 dB
nbar: 4

View: 3D Array Directivity

3D Directi
100 MHz N

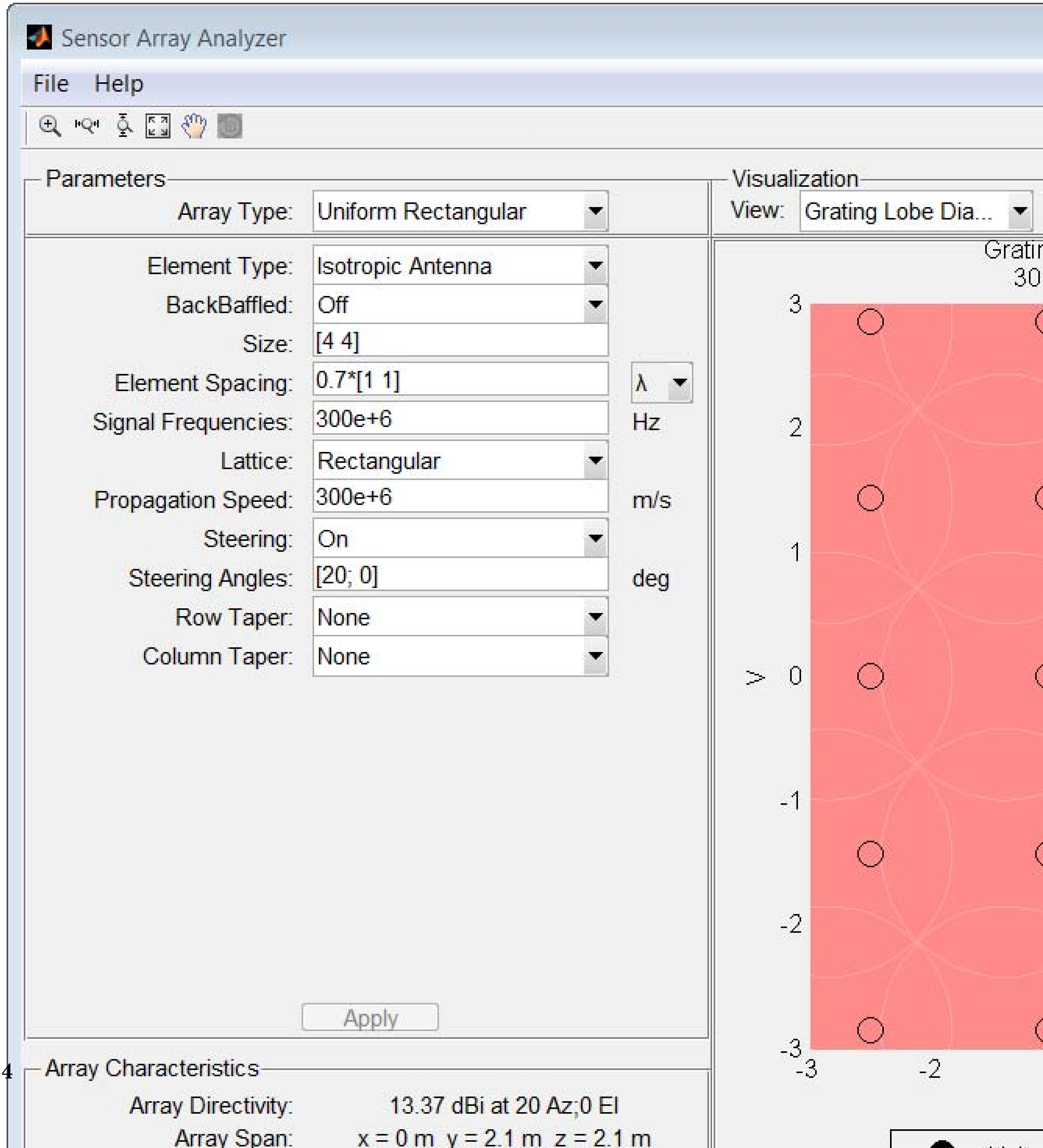
Array Directivity: 16.03 dBi at 0 Az;0 El
Array Span: x = 0 m y = 7.5 m z = 7.5 m

directivity. Without tapering, the array directivity for this URA is 17.2 dB. With tapering, the array directivity loses 1 dB to yield 16.0 dB.

Grating Lobes for a Rectangular Array

Show the grating lobe diagram of a 4-by-4 uniform rectangular array (URA) designed to detect and localize a 300 MHz signal. Set the **Array Type** to `Uniform Rectangular`, the **Element Type** to `Isotropic Antenna`, and the array **Size** to `[4 4]`. Set the **Signal Frequencies** to `300e+6`. By setting the row and column **Element Spacing** to 0.7 wavelengths, you create a spatially undersampled array.

This figure shows the grating lobe diagram produced when you beamform the array towards the angle [20,0]. The mainlobe is designated by the small black-filled circle. The multiple grating lobes are designated by the small unfilled black circles. The larger black circle is called the physical region, for which $u^2 + v^2 \leq 1$. The mainlobe always lies in the physical region. The grating lobes may or may not lie in the physical region. Any grating lobe in the physical region leads to an ambiguity in the direction of the incoming wave. The green region shows where the mainlobe can be pointed without any grating lobes appearing in the physical region. If the mainlobe is set to point outside the green region, a grating lobe moves into the physical region.


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Gratin
30
3
2
1
0
-1
-2
-3
v
-3
-2

View: Grating Lobe Dia...

Array Type: Uniform Rectangular

Element Type: Isotropic Antenna

BackBaffled: Off

Size: [4 4]

Element Spacing: 0.7*[1 1] λ

Signal Frequencies: 300e+6 Hz

Lattice: Rectangular

Propagation Speed: 300e+6 m/s

Steering: On

Steering Angles: [20; 0] deg

Row Taper: None

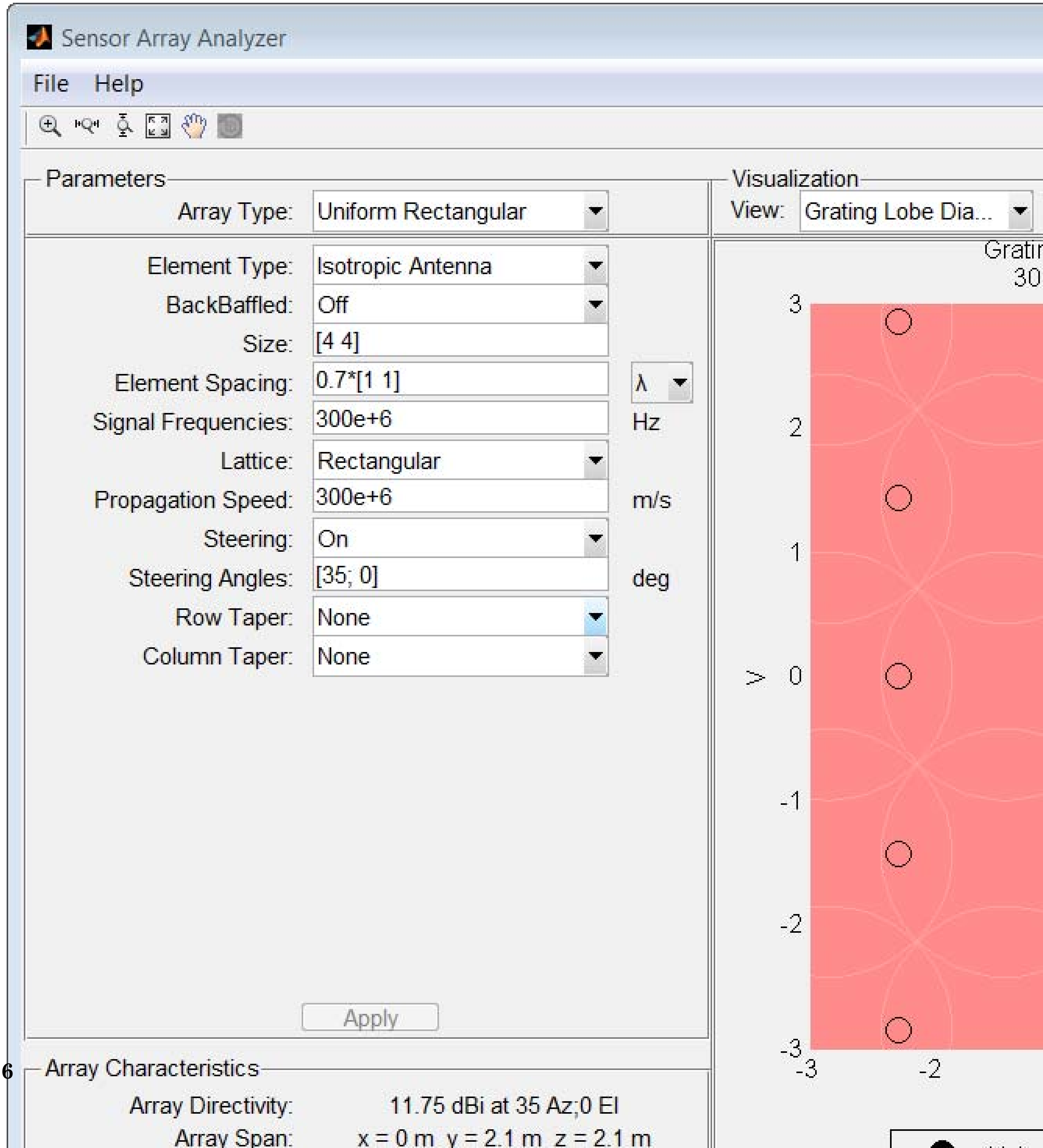
Column Taper: None

Array Characteristics

Array Directivity: 13.37 dBi at 20 Az;0 El

Array Span: x = 0 m y = 2.1 m z = 2.1 m

The next figure shows what happens when the pointing direction lies outside the green region. In this case, one grating lobe moves into the physical region.


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Array Characteristics
Gratir
30
3
2
1
0
-1
-2
-3
-3
-2
v

Array Type: Uniform Rectangular

Element Type: Isotropic Antenna

BackBaffled: Off

Size: [4 4]

Element Spacing: 0.7*[1 1] λ

Signal Frequencies: 300e+6 Hz

Lattice: Rectangular

Propagation Speed: 300e+6 m/s

Steering: On

Steering Angles: [35; 0] deg

Row Taper: None

Column Taper: None

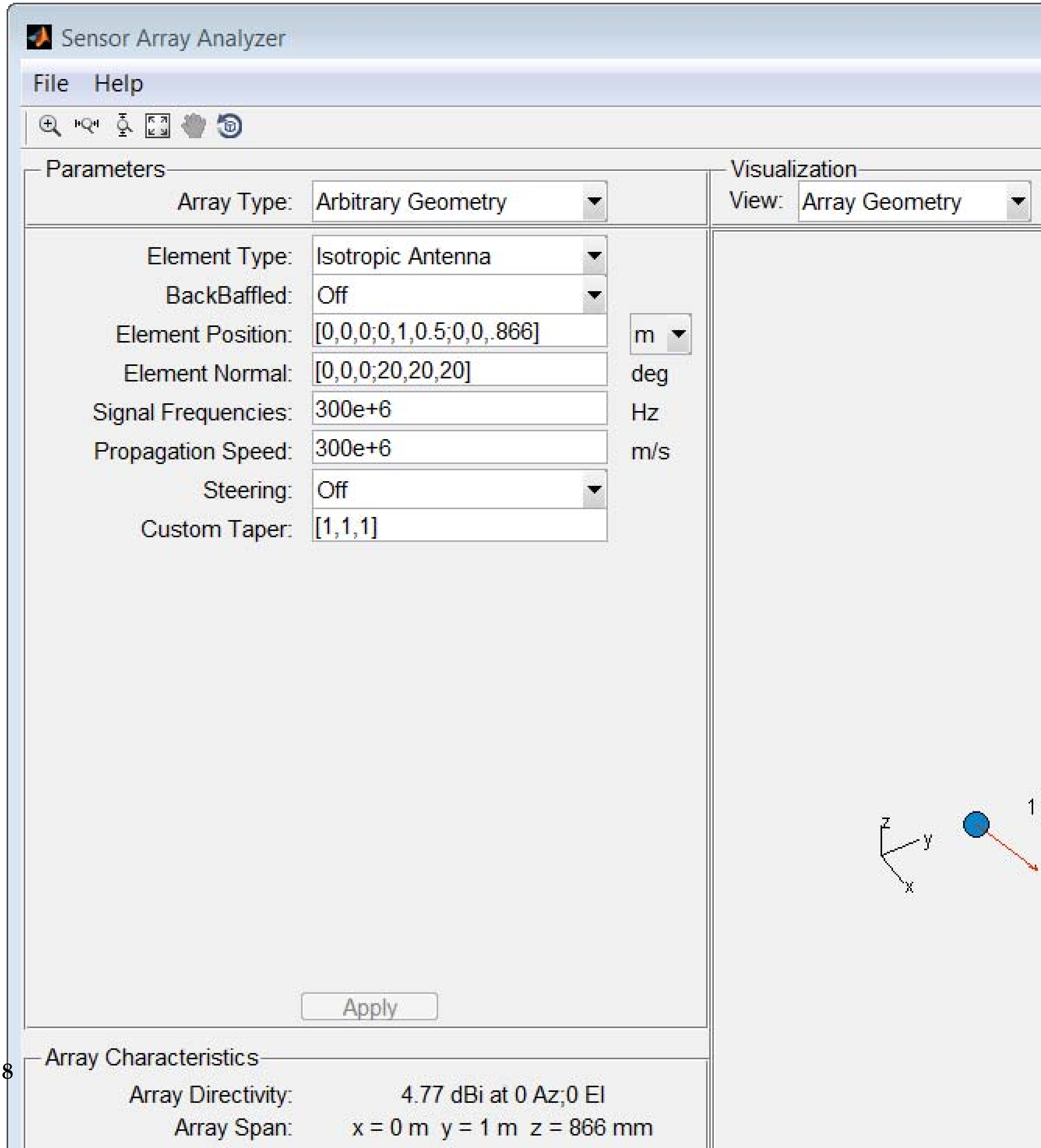
View: Grating Lobe Dia...

Array Directivity: 11.75 dBi at 35 Az;0 El

Array Span: x = 0 m y = 2.1 m z = 2.1 m

Specify arbitrary array geometry

You can specify an array which has an arbitrary placement of sensors. This simple example shows how to construct a triangular array of three isotropic antenna elements. The elements are placed at `[0,0,0]'`, `[0,1,0.5]'`, and `[0,0,0.866]'`. All elements have the same normal direction `[0,20]`, pointing to 0° in azimuth and 20° in elevation.



Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Array Characteristics
z
y
x
1

Array Type: Arbitrary Geometry

View: Array Geometry

Element Type: Isotropic Antenna

BackBaffled: Off

Element Position: [0,0,0;0,1,0.5;0,0,.866] m

Element Normal: [0,0,0;20,20,20] deg

Signal Frequencies: 300e+6 Hz

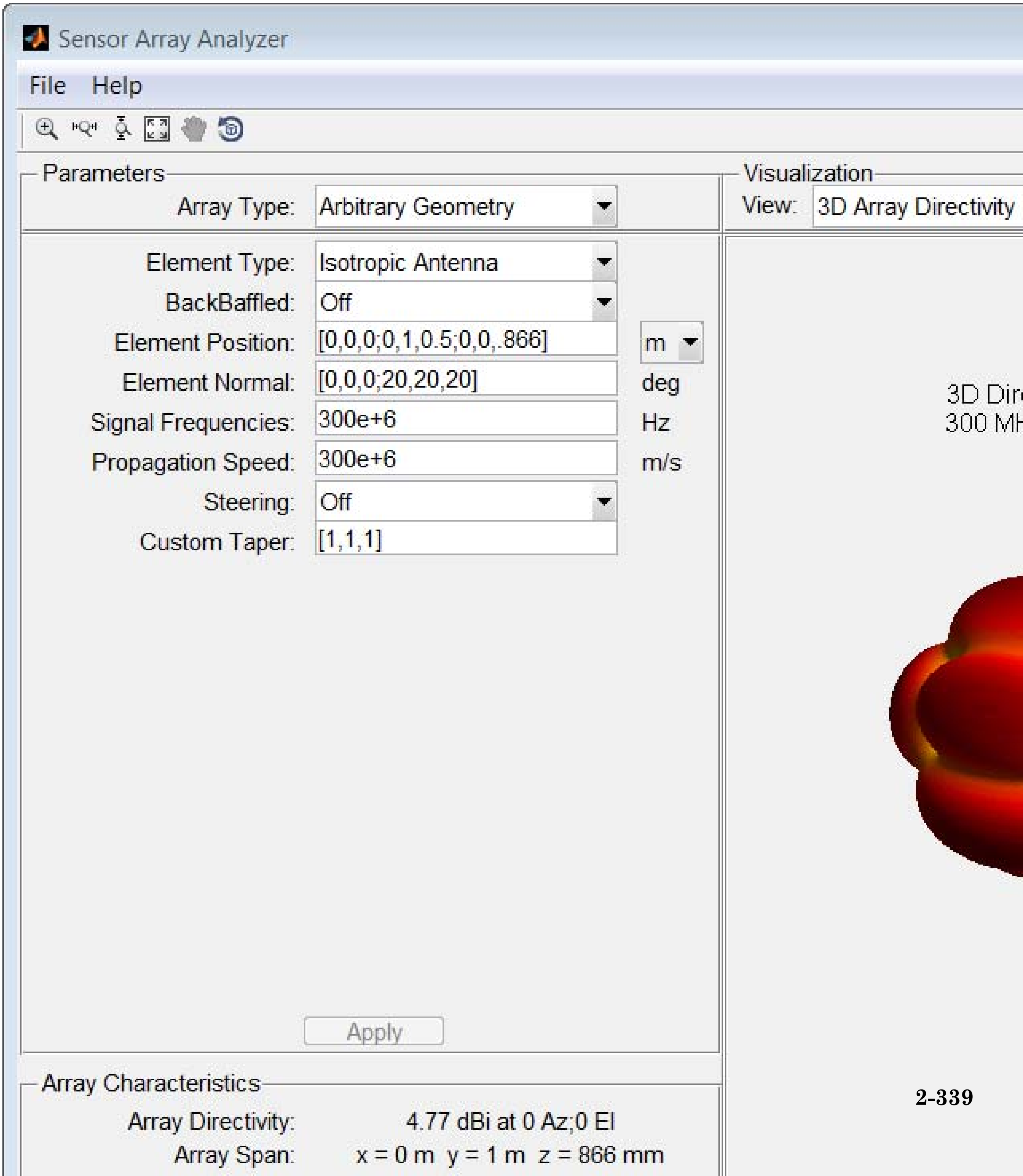
Propagation Speed: 300e+6 m/s

Steering: Off

Custom Taper: [1,1,1]

Array Directivity: 4.77 dBi at 0 Az;0 El

Array Span: x = 0 m y = 1 m z = 866 mm


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Array Characteristics
3D Dir
300 MH

Array Type: Arbitrary Geometry

View: 3D Array Directivity

Element Type: Isotropic Antenna

BackBaffled: Off

Element Position: [0,0,0;0,1,0.5;0,0,.866] m

Element Normal: [0,0,0;20,20,20] deg

Signal Frequencies: 300e+6 Hz

Propagation Speed: 300e+6 m/s

Steering: Off

Custom Taper: [1,1,1]

Array Directivity: 4.77 dBi at 0 Az;0 El

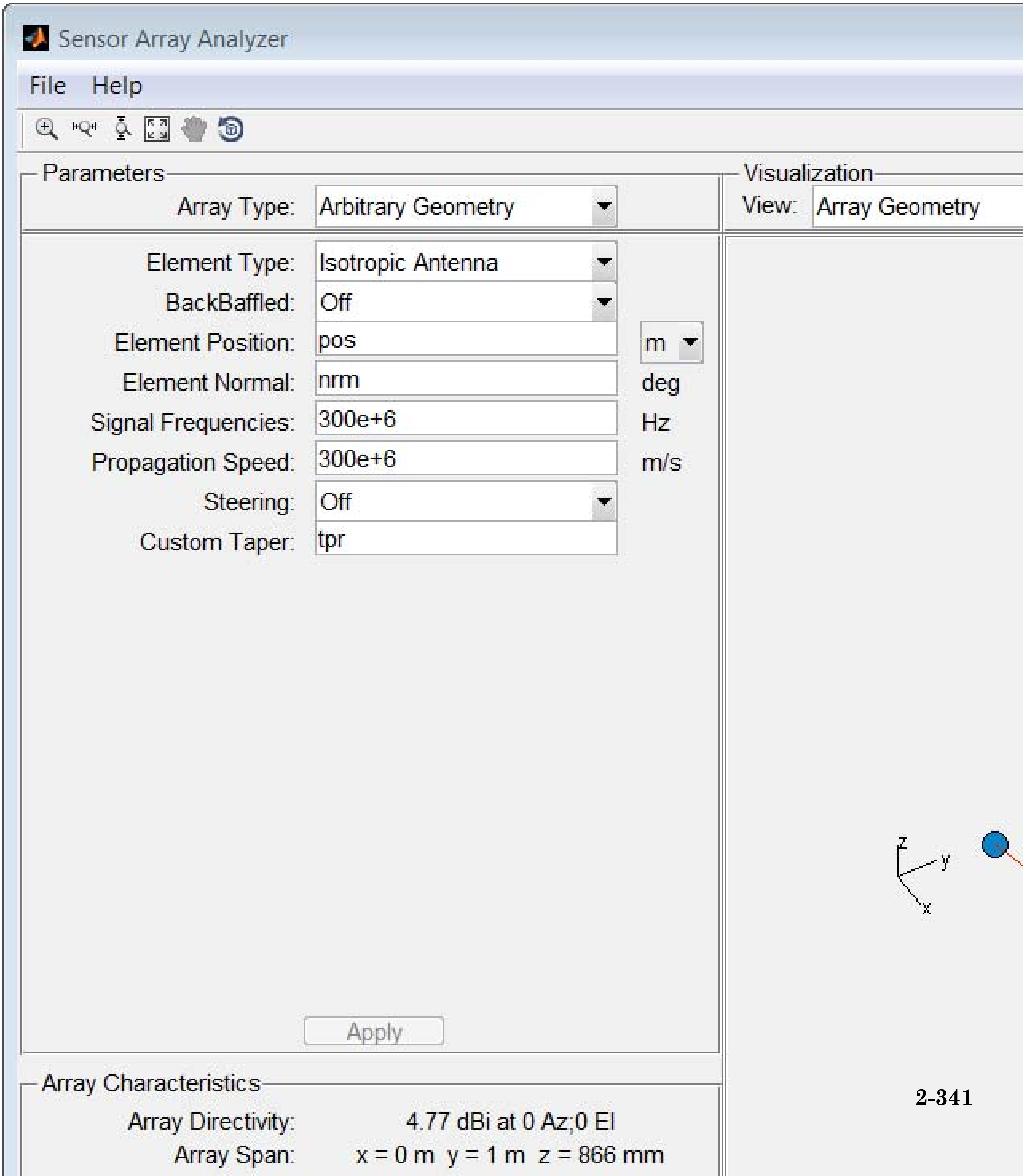
Array Span: x = 0 m y = 1 m z = 866 mm

Specify array geometry using variables

Specify an array which has an arbitrary placement of sensors but, in this case, create MATLAB variables or arrays at the command line and use them in the appropriate `sensorArrayAnalyzer` fields. This simple example shows the how to construct a triangular array of three isotropic antenna elements. At the command line, create an element position array, `pos`, an element normal array, `nrm`, and a taper value array, `tpr`.

```
pos = [0,0,0;0,1,0.5;0,0,0.866];
nrm = [0,0,0;20,20,20];
tpr = [1,1,1];
```

Then, enter these variables in the appropriate `sensorArrayAnalyzer` fields.


Sensor Array Analyzer
File
Help
Parameters
Visualization
Apply
Array Characteristics
z
y
x

Array Type: Arbitrary Geometry

View: Array Geometry

Element Type: Isotropic Antenna

BackBaffled: Off

Element Position: pos m

Element Normal: nrm deg

Signal Frequencies: 300e+6 Hz

Propagation Speed: 300e+6 m/s

Steering: Off

Custom Taper: tpr

Array Directivity: 4.77 dBi at 0 Az;0 El

Array Span: x = 0 m y = 1 m z = 866 mm

sensorcov

Purpose

Sensor spatial covariance matrix

Syntax

```
xcov = sensorcov(pos,ang)
xcov = sensorcov(pos,ang,ncov)
xcov = sensorcov(pos,ang,ncov,scov)
```

Description

`xcov = sensorcov(pos,ang)` returns the sensor spatial covariance matrix, `xcov`, for narrowband plane wave signals arriving at a sensor array. The sensor array is defined by the sensor positions specified in the `pos` argument. The signal arrival directions are specified by azimuth and elevation angles in the `ang` argument. In this syntax, the noise power is assumed to be zero at all sensors, and the signal power is assumed to be unity for all signals.

`xcov = sensorcov(pos,ang,ncov)` specifies, in addition, the spatial noise covariance matrix, `ncov`. This value represents the noise power on each sensor as well as the correlation of the noise between sensors. In this syntax, the signal power is assumed to be unity for all signals. This syntax can use any of the input arguments in the previous syntax.

`xcov = sensorcov(pos,ang,ncov,scov)` specifies, in addition, the signal covariance matrix, `scov`, which represents the power in each signal and the correlation between signals. This syntax can use any of the input arguments in the previous syntaxes.

Input Arguments

pos - Positions of array sensor elements

1-by-N real-valued vector | 2-by-N real-valued matrix | 3-by-N real-valued matrix

Positions of the elements of a sensor array specified as a 1-by-N vector, a 2-by-N matrix, or a 3-by-N matrix. In this vector or matrix, N represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. You define sensor position units in term of signal wavelength. If `pos` is a 1-by-N vector, then it represents the y-coordinate of the sensor elements of a line array. The x and z-coordinates are assumed to be zero. If `pos` is a 2-by-N matrix, then it represents the *(y,z)*-coordinates of the sensor elements of a planar array which is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. If `pos` is a 3-by-*N* matrix, then the array has arbitrary shape.

Example: [0, 0, 0; .1, .2, .3; 0,0,0]

Data Types
`double`

ang - Arrival directions of incoming signals

1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Arrival directions of incoming signals specified as a 1-by-*M* vector or a 2-by-*M* matrix, where *M* is the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the incoming signal `[az;el]`. Angular units are specified in degrees. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the arrival direction vector onto the *xy* plane. It is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the arrival direction vector and *xy*-plane. It is positive when measured towards the *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero.

Example: [45;0]

Data Types
`double`

ncov - Noise spatial covariance matrix

0 (default) | non-negative real-valued scalar | *1*-by-*N* non-negative real-valued vector | *N*-by-*N* positive definite, complex-valued matrix

Noise spatial covariance matrix specified as a non-negative, real-valued scalar, a non-negative, 1-by-*N* real-valued vector or an *N*-by-*N*, positive definite, complex-valued matrix. In this argument, *N* is the number of sensor elements. Using a non-negative scalar results in a noise spatial covariance matrix that has identical white noise power values (in watts) along its diagonal and has off-diagonal values of zero. Using a non-negative real-valued vector results in a noise spatial covariance that has diagonal values corresponding to the entries in `ncov` and has off-diagonal entries of zero. The diagonal entries represent the independent white noise power values (in watts) in each sensor. If `ncov` is *N*-by-*N* matrix, this value represents the full noise spatial covariance matrix between all sensor elements.

Example: [1,1,4,6]

Data Types
`double`
Complex Number Support: Yes

scov - Signal covariance matrix

1 (default) | non-negative real-valued scalar | *1*-by-*M* non-negative real-valued vector | *N*-by-*M* positive semidefinite, complex-valued matrix

Signal covariance matrix specified as a non-negative, real-valued scalar, a *1*-by-*M* non-negative, real-valued vector or an *M*-by-*M* positive semidefinite, matrix representing the covariance matrix between *M* signals. The number of signals is specified in `ang`. If `scov` is a nonnegative scalar, it assigns the same power (in watts) to all incoming signals which are assumed to be uncorrelated. If `scov` is a 1-by-*M* vector, it assigns the separate power values (in watts) to each incoming signal which are also assumed to be uncorrelated. If `scov` is an *M*-by-*M* matrix, then it represents the full covariance matrix between all incoming signals.

Example: [1 0 ; 0 2]

Data Types
`double`
Complex Number Support: Yes

Output Arguments

xcov - Sensor spatial covariance matrix

Complex-valued *N*-by-*N* matrix

Sensor spatial covariance matrix returned as a complex-valued, *N*-by-*N* matrix. In this matrix, *N* represents the number of sensor elements of the array.

Examples

Covariance Matrix for Two Signals without Noise

Create a covariance matrix for a 3-element, half-wavelength-spaced line array. Use the default syntax, which assumes no noise power and unit signal power.

```
N = 3;     % Elements in array
d = 0.5;    % sensor spacing half wavelength
elementPos = (0:N-1)*d;
xcov = sensorcov(elementPos,[30 60]);
```

```
xcov =

   2.0000 + 0.0000i  -0.9127 - 1.4086i  -0.3339 + 0.7458i
  -0.9127 + 1.4086i   2.0000 + 0.0000i  -0.9127 - 1.4086i
  -0.3339 - 0.7458i  -0.9127 + 1.4086i   2.0000 + 0.0000i
```

The diagonal terms represent the sum of the two signal powers.

Covariance Matrix for Two Independent Signals with 10 dB SNR

Create a spatial covariance matrix for a 3-element, half-wavelength-spaced line array. Assume there are two incoming unit-power signals and there is a noise value of –10 dB. By default, `scov` is the identity matrix.

```
  N = 3;     % Elements in array
  d = 0.5;    % sensor spacing half wavelength
  elementPos = (0:N-1)*d;
  xcov = sensorcov(elementPos,[30 35],db2pow(-10));
```

```
xcov =
   2.1000 + 0.0000i  -0.2291 - 1.9734i  -1.8950 + 0.4460i
  -0.2291 + 1.9734i   2.1000 + 0.0000i  -0.2291 - 1.9734i
  -1.8950 - 0.4460i  -0.2291 + 1.9734i   2.1000 + 0.0000i
```

The diagonal terms represent the two signal powers plus noise power at each sensor.

Covariance Matrix for Two Correlated Signals with 10 dB SNR

Compute the covariance matrix for a 3-element half-wavelength spaced line array when there is some correlation between two signals. The correlation can model, for example, multipath propagation caused by reflection from a surface. Assume a noise power value of –10 dB.

```
N = 3;     % Elements in array
d = 0.5;    % sensor spacing half wavelength
elementPos = (0:N-1)*d;
scov = [1, 0.8; 0.8, 1];
xcov = sensorcov(elementPos,[30 35],db2pow(-10),scov);
```

```
xcov =

   3.7000 + 0.0000i  -0.4124 - 3.5521i  -3.4111 + 0.8028i
  -0.4124 + 3.5521i   3.6574 + 0.0000i  -0.4026 - 3.4682i
  -3.4111 - 0.8028i  -0.4026 + 3.4682i   3.5321 + 0.0000i
```

References

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

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

See Also

`cbfweights` | `lcmvweights` | `mvdrweights` | `steervec` | `sensorsig` | `phased.SteeringVector`

sensorsig

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.

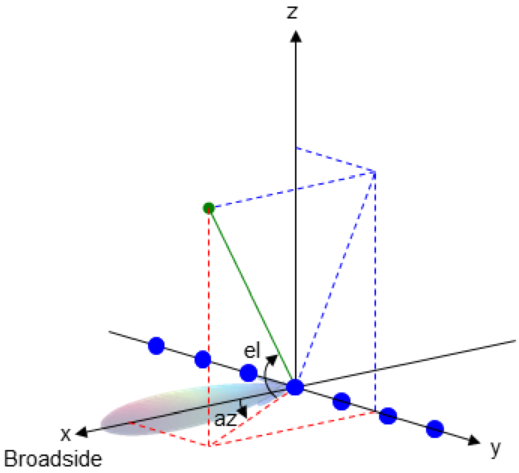
Definitions

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.

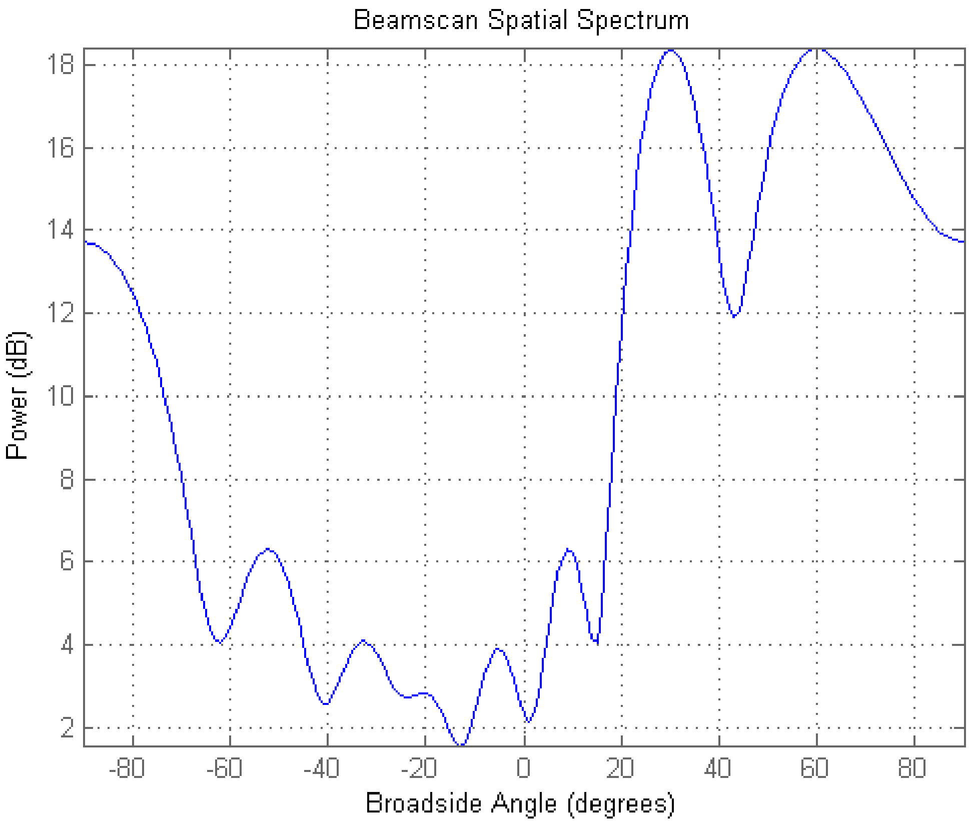
```
x = sensorsig(getElementPosition(ha)/lambda,...
   100,[30 60],db2pow(-10));
```

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.

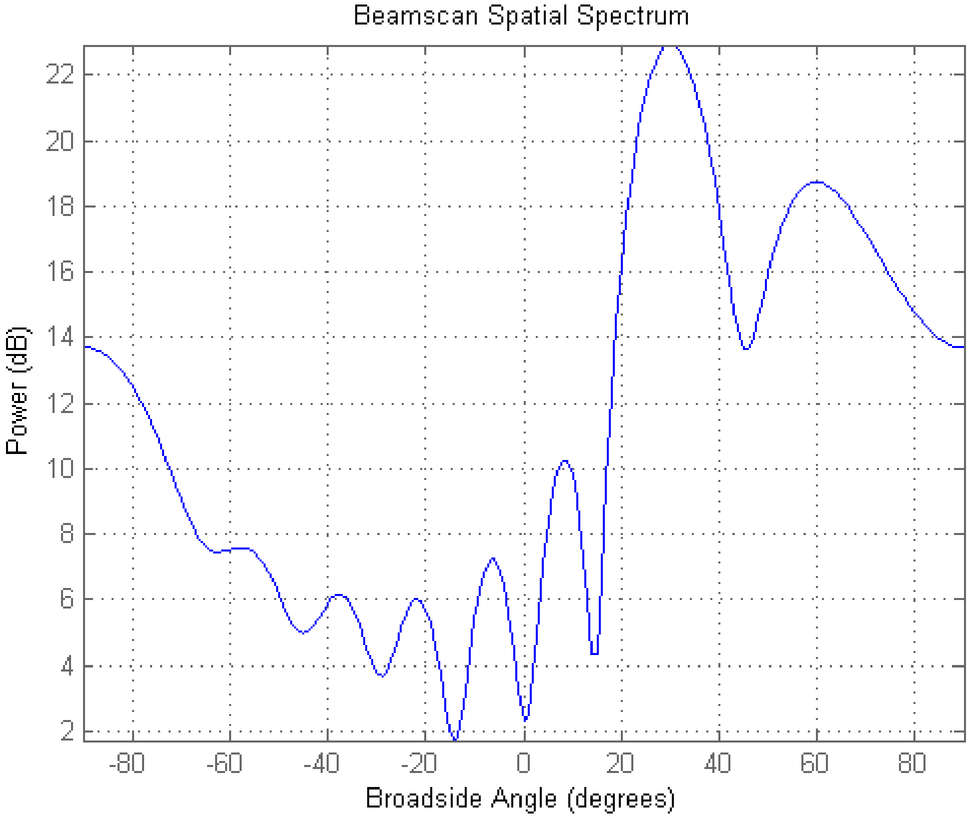
```
ang = [30 60];
scov = [3 1];
x = sensorsig(getElementPosition(ha)/lambda,...
   100,ang,db2pow(-10),scov);
```

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).

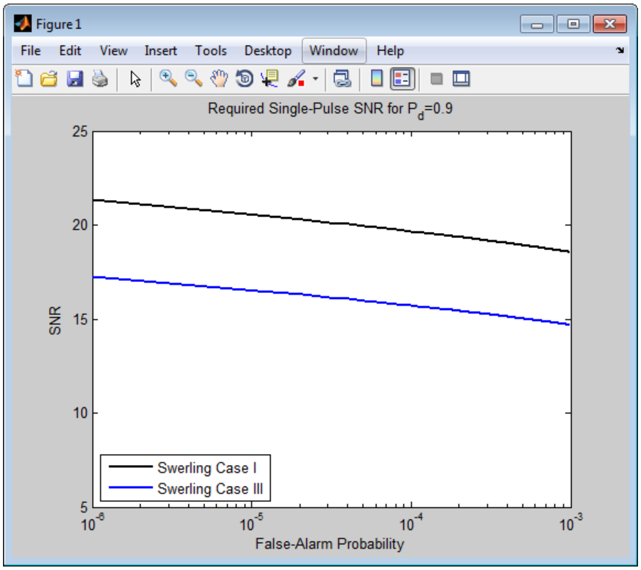
The Swerling case numbers consider all combinations of two decorrelation models (scan-to-scan; pulse-to-pulse) and two RCS PDFs (based on the presence or absence of a dominant scatterer).

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 vector from spherical basis components to Cartesian components

Syntax

`vr = sph2cartvec(vs,az,el)`

Description

`vr = sph2cartvec(vs,az,el)` converts the components of a vector or set of vectors, `vs`, from their *spherical basis representation* to their representation in a local Cartesian coordinate system. A spherical basis representation is the set of components of a vector projected into the right-handed spherical basis given by $(\mathbf{e}_{az}, \mathbf{e}_{el}, \mathbf{e}_R)$. The orientation of a spherical basis depends upon its location on the sphere as determined by azimuth, `az`, and elevation, `el`.

Input Arguments

vs - Vector in spherical basis representation

3-by-1 column vector | 3-by-N matrix

Vector in spherical basis representation specified as a 3-by-1 column vector or 3-by-N matrix. Each column of `vs` contains the three components of a vector in the right-handed spherical basis $(\mathbf{e}_{az}, \mathbf{e}_{el}, \mathbf{e}_R)$.

Example: `[4.0; -3.5; 6.3]`

Data Types
`double`
Complex Number Support: Yes

az - Azimuth angle

scalar in range [–180,180]

Azimuth angle specified as a scalar in the closed range [–180,180]. Angle units are in degrees. To define the azimuth angle of a point on a sphere, construct a vector from the origin to the point. The azimuth angle is the angle in the *xy*-plane from the positive *x*-axis to the vector's orthogonal projection into the *xy*-plane. As examples, zero azimuth angle and zero elevation angle specify a point on the *x*-axis while an azimuth angle of 90° and an elevation angle of zero specify a point on the *y*-axis.

Example: `45`

Data Types
`double`

el - Elevation angle

scalar in range [–90,90]

Elevation angle specified as a scalar in the closed range [–90,90]. Angle units are in degrees. To define the elevation of a point on the sphere, construct a vector from the origin to the point. The elevation angle is the angle from its orthogonal projection into the *xy*-plane to the vector itself. As examples, zero elevation angle defines the equator of the sphere and ±90° elevation define the north and south poles, respectively.

Example: `30`

Data Types
`double`

Output Arguments

vr - Vector in Cartesian representation

3-by-1 column vector | 3-by-N matrix

Cartesian vector returned as a 3-by-1 column vector or 3-by-N matrix having the same dimensions as `vs`. Each column of `vr` contains the three components of the vector in the right-handed *x,y,z* basis.

Examples

Cartesian Representation of Azimuthal Vector

Start with a vector in a spherical basis located at 45° azimuth, 45° elevation. The vector points along the azimuth direction. Compute its components with respect to Cartesian coordinates.

```
vs = [1;0;0];
vr = sph2cartvec(vs,45,45)
```

```
vr =

   -0.7071
    0.7071
```

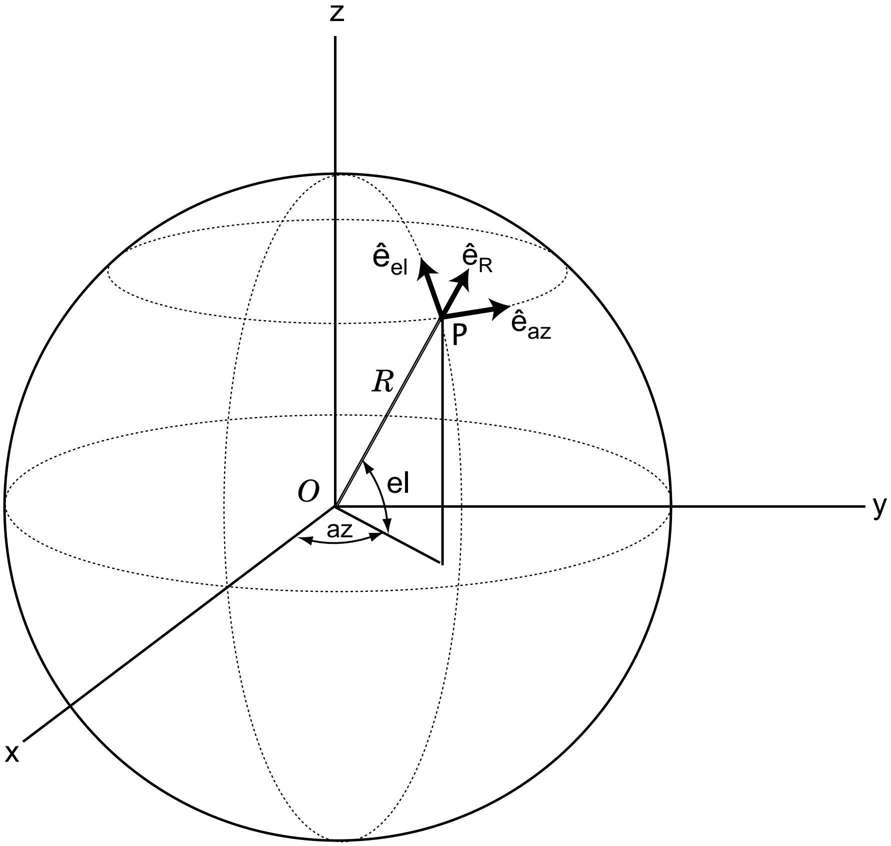
```
0
```

Definitions

Spherical basis representation of vectors

The spherical basis is a set of three mutually orthogonal unit vectors $(\mathbf{e}_{az}, \mathbf{e}_{el}, \mathbf{e}_{R})$ defined at a point on the sphere. The first unit vector points along lines of azimuth at constant radius and elevation. The second points along the lines of elevation at constant azimuth and radius. Both are tangent to the surface of the sphere. The third unit vector points radially outward.

The orientation of the basis changes from point to point on the sphere but is independent of R so as you move out along the radius, the basis orientation stays the same. The following figure illustrates the orientation of the spherical basis vectors as a function of azimuth and elevation:



For any point on the sphere specified by *az* and *el*, the basis vectors are given by:

$$\begin{aligned}
\hat{\mathbf{e}}_{\mathbf{az}} &= -\sin(az)\hat{\mathbf{i}} + \cos(az)\hat{\mathbf{j}} \\
\hat{\mathbf{e}}_{\mathbf{el}} &= -\sin(el)\cos(az)\hat{\mathbf{i}} - \sin(el)\sin(az)\hat{\mathbf{j}} + \cos(el)\hat{\mathbf{k}} \\
\hat{\mathbf{e}}_{\mathbf{R}} &= \cos(el)\cos(az)\hat{\mathbf{i}} + \cos(el)\sin(az)\hat{\mathbf{j}} + \sin(el)\hat{\mathbf{k}} \quad .
\end{aligned}$$

Any vector can be written in terms of components in this basis as $\mathbf{v} = v_{az}\hat{\mathbf{e}}_{\mathbf{az}} + v_{el}\hat{\mathbf{e}}_{\mathbf{el}} + v_R\hat{\mathbf{e}}_{\mathbf{R}}$. The transformations between spherical basis components and Cartesian components take the form

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} -\sin(az) & -\sin(el)\cos(az) & \cos(el)\cos(az) \\ \cos(az) & -\sin(el)\sin(az) & \cos(el)\sin(az) \\ 0 & \cos(el) & \sin(el) \end{bmatrix} \begin{bmatrix} v_{az} \\ v_{el} \\ v_R \end{bmatrix}$$

.

and

$$\begin{bmatrix} v_{az} \\ v_{el} \\ v_R \end{bmatrix} = \begin{bmatrix} -\sin(az) & \cos(az) & 0 \\ -\sin(el)\cos(az) & -\sin(el)\sin(az) & \cos(el) \\ \cos(el)\cos(az) & \cos(el)\sin(az) & \sin(el) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}.$$

See Also

`azelaxes` | `cart2sphvec`

spsmooth

Purpose

Spatial smoothing

Syntax

```
RSM = spsmooth(R,L)
RSM = spsmooth(R,L,'fb')
```

Description

`RSM = spsmooth(R,L)` computes an averaged spatial covariance matrix, `RSM`, from the full spatial covariance matrix, `R`, using *spatial smoothing* (see Van Trees [1], p. 605). Spatial smoothing creates a smaller averaged covariance matrix over L maximum overlapped subarrays. L is a positive integer less than N. The resulting covariance matrix, `RSM`, has dimensions (N–L+1)-by-(N–L+1). Spatial smoothing is useful when two or more signals are correlated.

`RSM = spsmooth(R,L,'fb')` computes an averaged covariance matrix and at the same time performing *forward-backward averaging*. This syntax can use any of the input arguments in the previous syntax.

Input Arguments

`R` - Spatial covariance matrix

Complex-valued positive-definite N-by-N matrix.

Spatial covariance matrix, specified as a complex-valued, positive-definite N-by-N matrix. In this matrix, N represents the number of sensor elements.

Example: [4.3162, –0.2777 –0.2337i; –0.2777 + 0.2337i , 4.3162]

Data Types
`double`
Complex Number Support: Yes

`L` - Maximum number of overlapped subarrays

Positive integer

Maximum number of overlapped subarrays, specified as a positive integer. The value L must be less than the number of sensors, N.

Example: 2

Data Types

`double`

Output Arguments

RSM - Smoothed covariance matrix

Complex-valued M-by-M matrix

Smoothed covariance matrix, returned as a complex-valued, M-by-M matrix. The dimension M is given by $M = N–L+1$.

Examples

Comparison of Smoothed and Nonsmoothed Covariance Matrices

Construct a 10-element half-wavelength-spaced uniform line array receiving two plane waves arriving from 0° and –25° azimuth. Both elevation angles are 0°. Assume the two signals are partially correlated. The SNR for each signal is 5 dB. The noise is spatially and temporally Gaussian white noise. First, create the spatial covariance matrix from the signal and noise. Then, solve for the number of signals, using `rootmusicdoa`. Next, perform spatial smoothing on the covariance matrix, using `spsmooth`, and solve for the signal arrival angles, again using `rootmusicdoa`.

Set up the array and signals. Then, generate the spatial covariance matrix for the array from the signals and noise.

```
N = 10;
d = 0.5;
elementPos = (0:N-1)*d;
angles = [0 -25];
ac = [1 1/5];
scov = ac'*ac;
R = sensorcov(elementPos,angles,db2pow(-5),scov);
```

Solve for the arrival angles using the original covariance matrix.

```
Nsig = 2;
doa =  rootmusicdoa(R,Nsig)
```

```
doa =
    0.3062   48.6810
```

The solved-for arrival angles are clearly wrong – they do not agree with the known angles of arrival used to create the covariance matrix.

Next, solve for the arrival angles using the smoothed covariance matrix.

```
Nsig = 2;
L = 2;
RSM = spsmooth(R,L);
doasm = rootmusicdoa(RSM,Nsig)

doasm =

   -25.0000  -0.0000
```

This time they do agree with the known angles of arrival.

References

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

See Also

`aictest` **|** `espritdoa` **|** `mdltest` **|** `rootmusicdoa`

Purpose

Steering vector

Syntax

`sv = steervec(pos,ang)`

Description

`sv = steervec(pos,ang)` returns the steering vector `sv` for each incoming plane wave or set of plane waves impinging on a sensor array. The steering vector represents the set of phase-delays for an incoming wave at each sensor element. The array is defined by its sensor element positions contained in the `pos` argument. The incoming wave arrival directions are specified by their azimuth and elevation angles in the `ang` argument. The steering vector, `sv`, is an *N*-by-*M* matrix. In this matrix, *N* represents the number of element positions in the sensor array while *M* represents the number of incoming waves. Each column of `sv` contains the steering vector for the corresponding direction specified in `ang`. All elements in the sensor array are assumed to be isotropic.

Input Arguments

pos - Positions of array sensor elements

1-by-*N* real-valued vector | 2-by-*N* real-valued matrix | 3-by-*N* real-valued matrix

Positions of the elements of a sensor array specified as a 1-by-*N* vector, a 2-by-*N* matrix, or a 3-by-*N* matrix. In this vector or matrix, *N* represents the number of elements of the array. Each column of `pos` represents the coordinates of an element. You define sensor position units in term of signal wavelength. If `pos` is a 1-by-*N* vector, then it represents the *y*-coordinate of the sensor elements of a line array. The *x* and *z*-coordinates are assumed to be zero. If `pos` is a 2-by-*N* matrix, then it represents the *(y,z)*-coordinates of the sensor elements of a planar array which is assumed to lie in the *yz*-plane. The *x*-coordinates are assumed to be zero. If `pos` is a 3-by-*N* matrix, then the array has arbitrary shape.

Example: [0, 0, 0; .1, .2, .3; 0,0,0]

Data Types
`double`

ang - Arrival directions of incoming signals

1-by-*M* real-valued vector | 2-by-*M* real-valued matrix

Arrival directions of incoming signals specified as a 1-by-*M* vector or a 2-by-*M* matrix, where *M* is the number of incoming signals. If `ang` is a 2-by-*M* matrix, each column specifies the direction in azimuth and elevation of the incoming signal `[az;el]`. Angular units are specified in degrees. The azimuth angle must lie between –180° and 180° and the elevation angle must lie between –90° and 90°. The azimuth angle is the angle between the *x*-axis and the projection of the arrival direction vector onto the *xy* plane. It is positive when measured from the *x*-axis toward the *y*-axis. The elevation angle is the angle between the arrival direction vector and *xy*-plane. It is positive when measured towards the *z* axis. If `ang` is a 1-by-*M* vector, then it represents a set of azimuth angles with the elevation angles assumed to be zero.

Example: [45;0]

Data Types
`double`

Output Arguments

sv - Steering vector

N-by-*M* complex-valued matrix

Steering vector returned as an *N*-by-*M* complex-valued matrix. In this matrix, *N* represents the number of sensor elements of the array and *M* represents the number of incoming plane waves. Each column of `sv` corresponds to a different entry in `ang`.

Examples

Steering Vector for a Short Line-array

Specify a line array of five elements spaced 10 cm apart. Then, specify an incoming plane wave with a frequency of 1 GHz and an arrival direction of 45° azimuth and 0° elevation. Compute the steering vector of this wave.

```
elementPos = (0:.1:.4); % meters
c = physconst('LightSpeed'); % speed of light;
fc = 1e9; % frequency
lam = c/fc; % wavelength
```

```
ang = [45;0]; % direction of arrive
sv = steervec(elementPos/lam,ang)

sv =

   1.0000 + 0.0000i
   0.0887 + 0.9961i
  -0.9843 + 0.1767i
  -0.2633 - 0.9647i
   0.9376 - 0.3478i
```

References

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

[2] Johnson, Don H. and D. Dudgeon. *Array Signal Processing.* Englewood Cliffs, NJ: Prentice Hall, 1993.

[3] Van Veen, B.D. and K. M. Buckley. “Beamforming: A versatile approach to spatial filtering”. *IEEE ASSP Magazine*, Vol. 5 No. 2 pp. 4–24.

See Also

`cbfweights` **|** `lcmvweights` **|** `mvdrweights` **|** `sensorcov` **|** `phased.SteeringVector`

Purpose

Stokes parameters of polarized field

Syntax

```
G = stokes(fv)
stokes(fv)
```

Description

`G = stokes(fv)` returns the four *Stokes* parameters `G` of a polarized field or set of fields specified in `fv`. The field should be expressed in terms of linear polarization components. The expression of a field in terms of a two-row vector of linear polarization components is called the *Jones vector formalism*.

`stokes(fv)` displays the Stokes parameters corresponding to `fv` as points on the *Poincare* sphere.

Input Arguments

fv - Field vector in linear polarization representation or linear polarization ratio

1-by-N complex-value row vector or 2-by-N complex-value matrix

Field vector in its linear polarization representation specified as a 2-by-N complex-valued matrix or in its linear polarization ratio representation specified as a 1-by-N complex-valued row vector. If `fv` is a matrix, each column of `fv` represents a field in the form `[Eh;Ev]`, where `Eh` and `Ev` are its horizontal and vertical linear polarization components. The expression of a field in terms of a two-row vector of linear polarization components is called the *Jones vector formalism*. If `fv` is a vector, each entry in `fv` is contains the polarization ratio, `Ev/Eh`.

Example: [sqrt(2)/2*1i; 1]

Data Types

`double`

Complex Number Support: Yes

Output Arguments

G - Stokes parameters

4-by-N matrix of Stokes parameters.

`G` contains the four Stokes parameters for each polarized field specified in `fv`. The Stokes parameters are computed from combinations of intensities of the field:

- G_0 describes the total intensity of the field.
- G_1 describes the preponderance of horizontal linear polarization intensity over vertical linear polarization intensity.
- G_2 describes the preponderance of +45° linear polarization intensity over -45° linear polarization intensity.
- G_3 describes the preponderance of right circular polarization intensity over left circular polarization intensity.

Examples

Stokes Vector

Create a left circularly-polarized field. Convert it to a linear representation and compute the Stokes vector.

```
cfv = [2;0];
fv = circpol2pol(cfv);
G=stokes(fv)
```

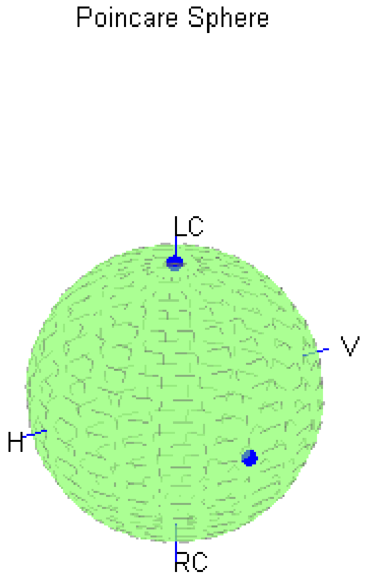
```
G =

    4.0000
         0
         0
    4.0000
```

Poincare Sphere

Display points on the Poincare sphere for a left circularly-polarized field and a 45 degree linear polarized field.

```
fv = [sqrt(2)/2, 1; sqrt(2)/2*1i, 1];
G = stokes(fv)
```

```
stokes(fv);


G =

    1.0000    2.0000
         0         0
         0    2.0000
    1.0000         0
```



The point at the north pole represents the left circularly-polarized field. The point on the equator represents the 45 degree linear polarized field.

References

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

[2] Jackson, J.D. , *Classical Electrodynamics*, 3rd Edition, John Wiley & Sons, 1998, pp. 299–302.

[3] Born, M. and E. Wolf, *Principles of Optics*, 7th Edition, Cambridge: Cambridge University Press, 1999, pp 25–32.

See Also

`circpol2pol` | `pol2circpol` | `polellip` | `polratio`

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');
```

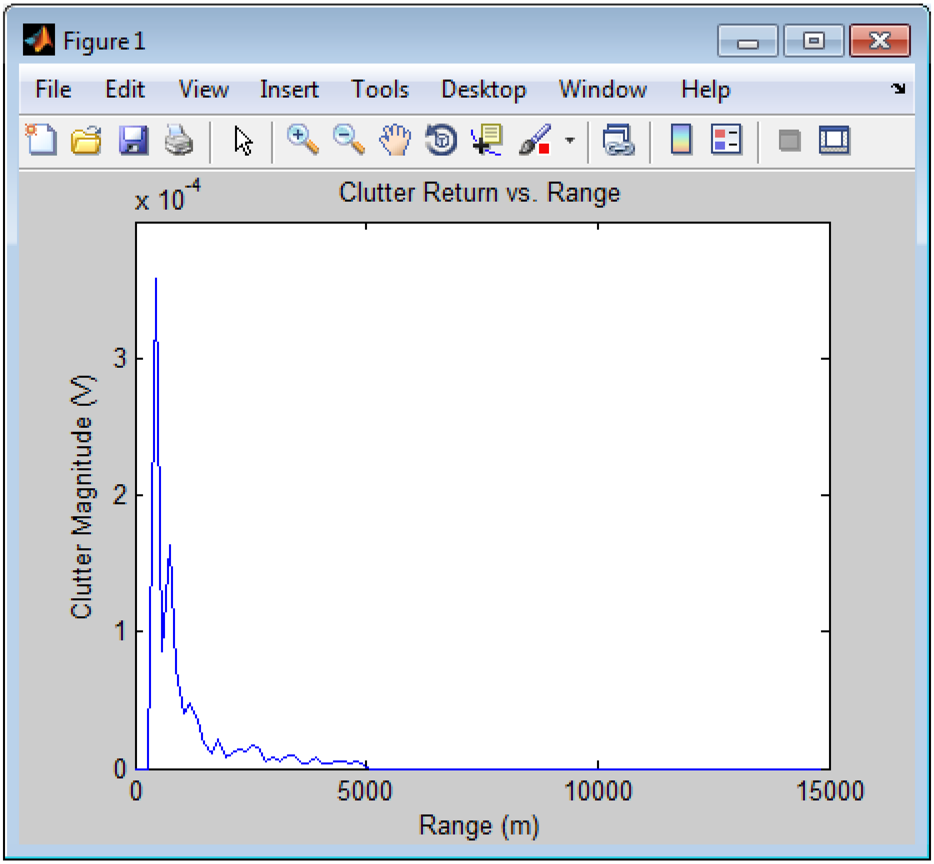


Figure 1
File
Edit
View
Insert
Tools
Desktop
Window
Help
Clutter Return vs. Range
x 10^{-4}
Clutter Magnitude (V)
0
1
2
3
0
5000
10000
15000
Range (m)

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`

unigrid

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.

In terms of azimuth and elevation, the *u* and *v* coordinates are

$u = \cos(\mathrm{el})\ \sin(\mathrm{az})$

$v = \sin(\mathrm{el})$

The values of u and v satisfy the inequalities

$-1 \leq u \leq 1$

$-1 \leq v \leq 1$

$u^2 + v^2 \leq 1$

Conversely, the phi and theta angles can be written in terms of u and v

$\tan(\varphi) = v/u$

$\sin(\theta) = \mathrm{sqrt}(u^2 + v^2)$

The azimuth and elevation angles can also be written in terms of u and v

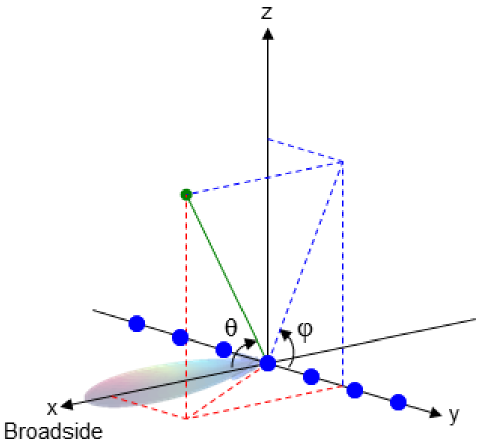
$\sin(\mathrm{el}) = v$

$\tan(\mathrm{az}) = u/\mathrm{sqrt}(1 - u^2 - v^2)$

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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi\sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi\tan\theta$$

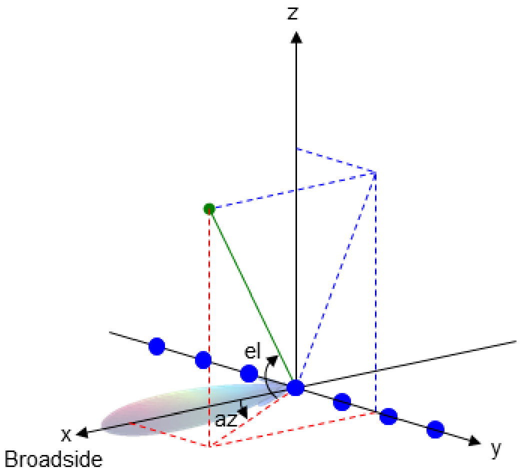
$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el})/\sin(\mathrm{az})$$

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 and v coordinates are the direction cosines of a vector with respect to the y-axis and z-axis, respectively.

The u/v coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, 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.

In terms of azimuth and elevation, the u and v coordinates are

$$u = \cos(\mathrm{el})\sin(\mathrm{az})$$

$$v = \sin(\mathrm{el})$$

The values of u and v satisfy the inequalities

$$-1 \leq u \leq 1$$

$-1 \leq v \leq 1$

$u^2 + v^2 \leq 1$

Conversely, the phi and theta angles can be written in terms of *u* and *v*

$\tan(\varphi) = v/u$

$\sin(\theta) = \mathrm{sqrt}(u^2 + v^2)$

The azimuth and elevation angles can also be written in terms of *u* and *v*

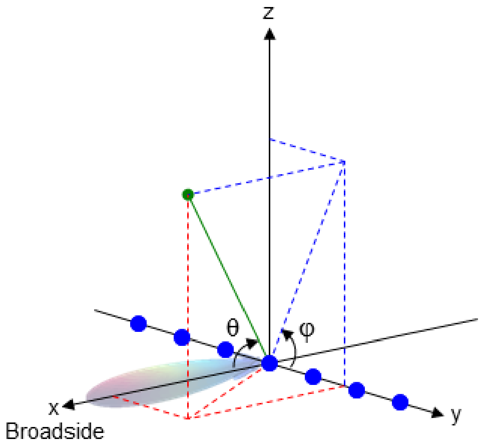
$\sin(\mathrm{el}) = v$

$\tan(\mathrm{az}) = u/\mathrm{sqrt}(1 - u^2 - v^2)$

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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi\sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi\tan\theta$$

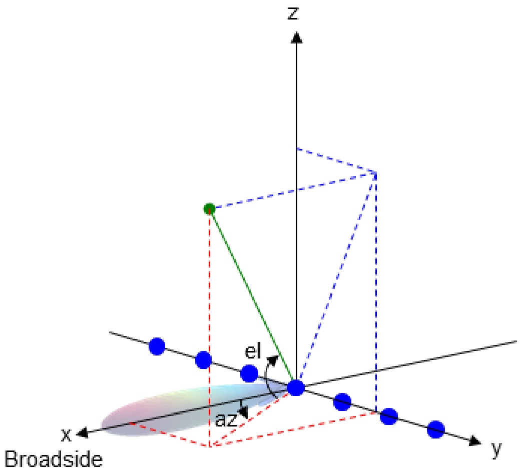
$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el})/\sin(\mathrm{az})$$

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 Converted Radiation Pattern

Convert a radiation pattern to azimuth/elevation form, with the angles spaced one degree apart.

Define the pattern in terms of u and v. For values outside the unit circle, u and v are undefined and the pattern value is 0.

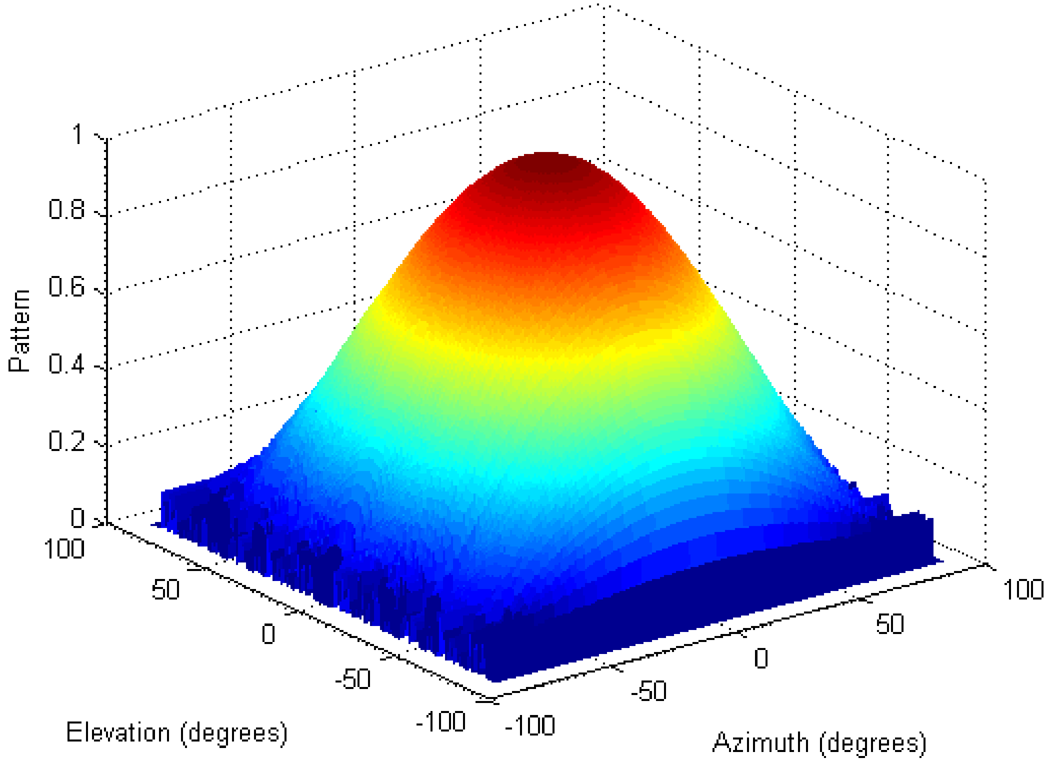
```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Convert the pattern to 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');
```



Convert Radiation Pattern Using Specific Azimuth/Elevation Values

Convert a radiation pattern to azimuth/elevation form, with the angles spaced five degrees apart.

Define the pattern in terms of u and v . For values outside the unit circle, u and v are undefined and the pattern value is 0.

```
u = -1:0.01:1;
```

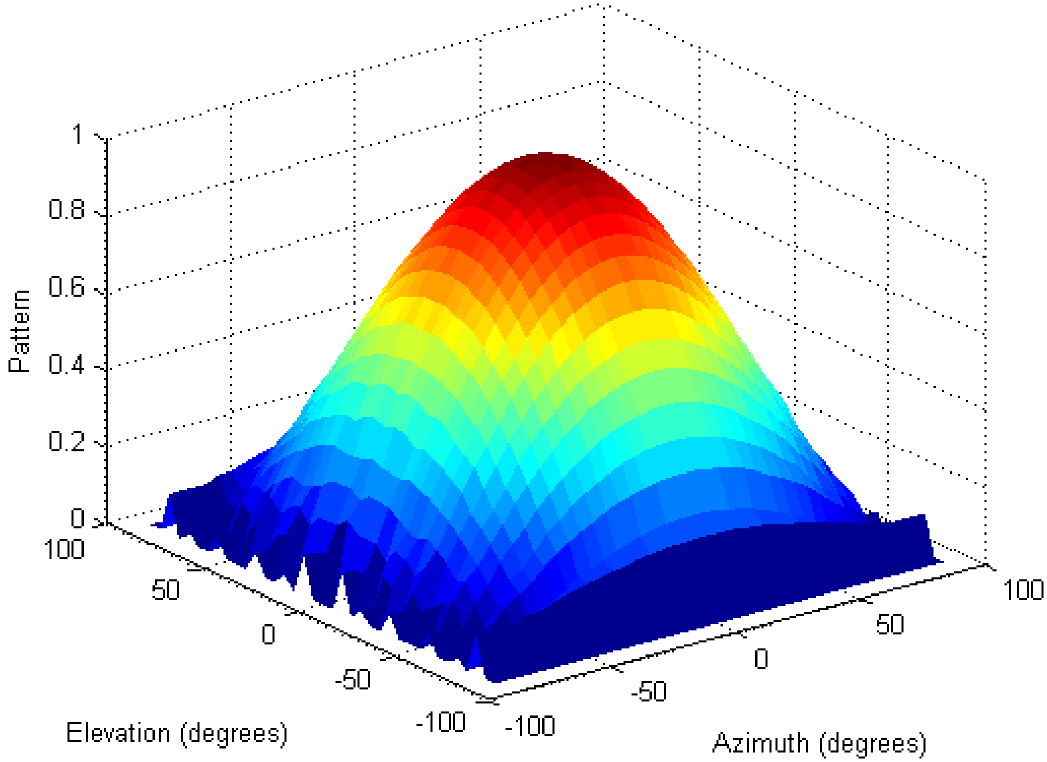
```
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Define the set of 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"

uv2phitheta

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.

In terms of azimuth and elevation, the *u* and *v* coordinates are

$u = \cos(\mathrm{el})\,\sin(\mathrm{az})$

$v = \sin(\mathrm{el})$

The values of u and v satisfy the inequalities

$-1 \le u \le 1$

$-1 \le v \le 1$

$u^2 + v^2 \le 1$

Conversely, the phi and theta angles can be written in terms of u and v

$\tan(\varphi) = v/u$

$\sin(\theta) = \mathrm{sqrt}(u^2 + v^2)$

The azimuth and elevation angles can also be written in terms of u and v

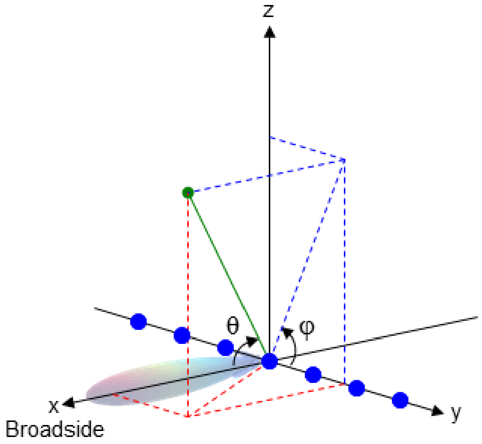
$\sin(\mathrm{el}) = v$

$\tan(\mathrm{az}) = u/\mathrm{sqrt}(1 - u^2 - v^2)$

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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi\sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi\tan\theta$$

$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el}) / \sin(\mathrm{az})$$

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* and *v* coordinates are the direction cosines of a vector with respect to the *y*-axis and *z*-axis, respectively.

The *u*/*v* coordinates for the hemisphere $x \geq 0$ are derived from the phi and theta angles, 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.

In terms of azimuth and elevation, the *u* and *v* coordinates are

$$u = \cos(\mathrm{el})\sin(\mathrm{az})$$

$$v = \sin(\mathrm{el})$$

The values of *u* and *v* satisfy the inequalities

$$-1 \leq u \leq 1$$

$$-1 \leq v \leq 1$$

$$u^2 + v^2 \leq 1$$

Conversely, the phi and theta angles can be written in terms of u and v

$\tan(\varphi) = v/u$

$\sin(\theta) = \mathrm{sqrt}(u^2 + v^2)$

The azimuth and elevation angles can also be written in terms of u and v

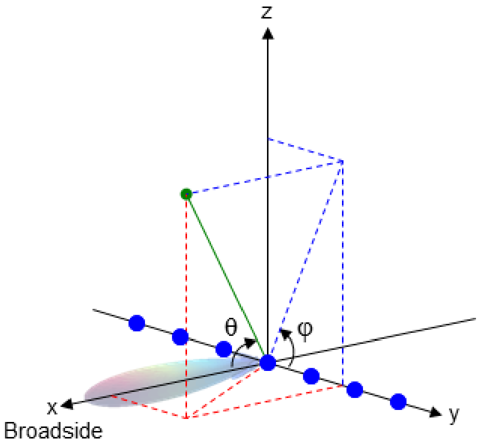
$\sin(el) = v$

$\tan(az) = u/\mathrm{sqrt}(1 - u^2 - v^2)$

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.



The coordinate transformations between φ/θ and *az/el* are described by the following equations

$$\sin(\mathrm{el}) = \sin\phi\sin\theta$$
$$\tan(\mathrm{az}) = \cos\phi\tan\theta$$

$$\cos\theta = \cos(\mathrm{el})\cos(\mathrm{az})$$
$$\tan\phi = \tan(\mathrm{el})/\sin(\mathrm{az})$$

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 Converted Radiation Pattern

Convert a radiation pattern to $\phi - \theta$ space with the angles spaced one degree apart.

Define the pattern in terms of u and v . For values outside the unit circle, u and v are undefined, and the pattern value is 0.

```
u = -1:0.01:1;
v = -1:0.01:1;
[u_grid,v_grid] = meshgrid(u,v);
```

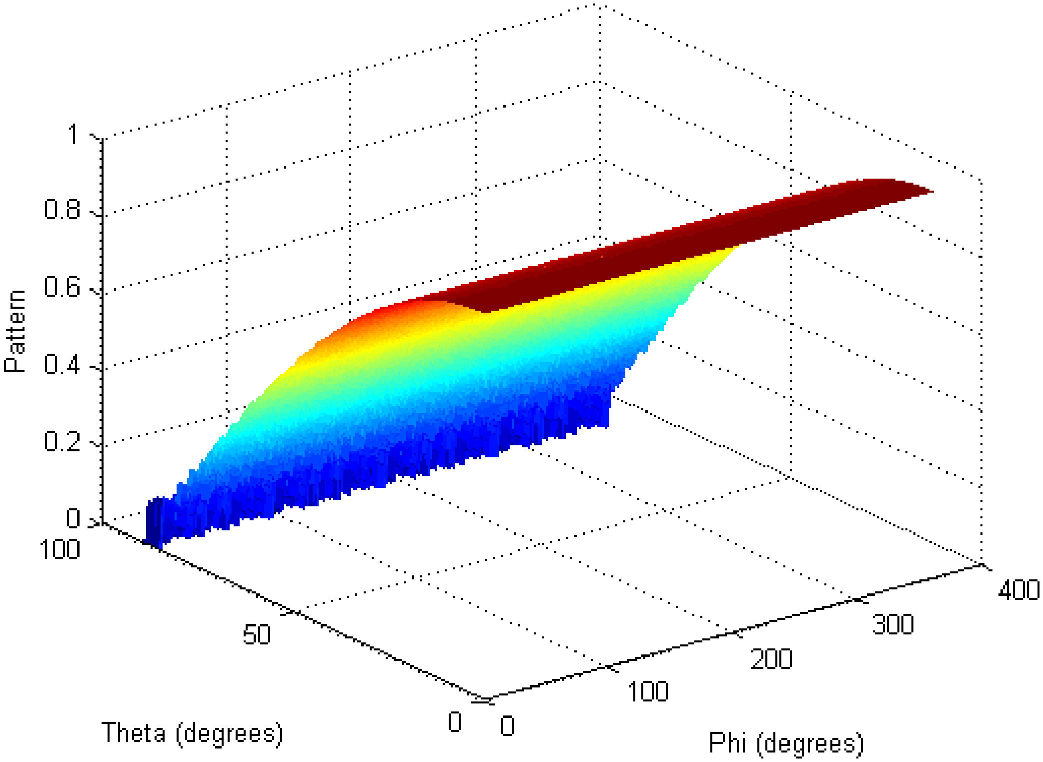
```
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Convert the pattern to $\phi - \theta$ space. Store the ϕ and θ angles for use in plotting.

```
[pat_phitheta,phi,theta] = uv2phithetapat(pat_uv,u,v);
```

Plot the result.

```
H = surf(phi,theta,pat_phitheta);
set(H,'LineStyle','none')
xlabel('Phi (degrees)');
ylabel('Theta (degrees)');
zlabel('Pattern');
```



Convert Radiation Pattern Using Specific Phi/Theta Values

Convert a radiation pattern to $\phi - \theta$ space with the angles spaced five degrees apart.

Define the pattern in terms of u and v. For values outside the unit circle, u and v are undefined, and the pattern value is 0.

```
u = -1:0.01:1;
v = -1:0.01:1;
```

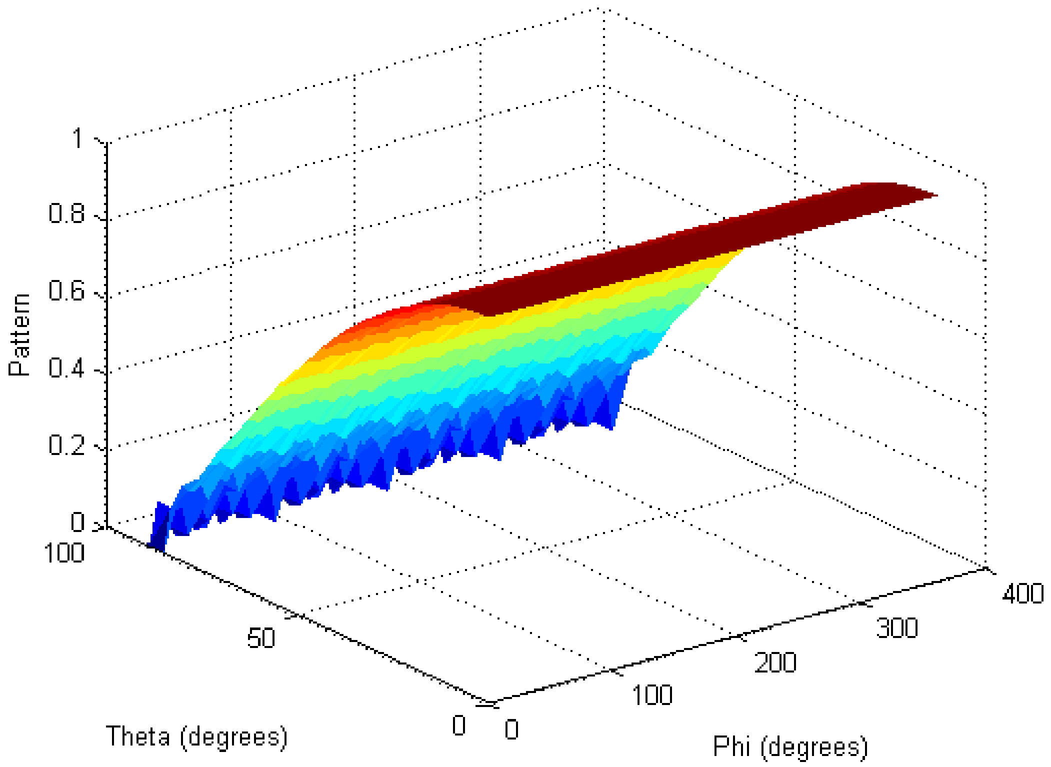
```
[u_grid,v_grid] = meshgrid(u,v);
pat_uv = sqrt(1 - u_grid.^2 - v_grid.^2);
pat_uv(hypot(u_grid,v_grid) >= 1) = 0;
```

Define the set of ϕ and θ angles at which to sample the pattern. Then, convert the pattern.

```
phi = 0:5:360;
theta = 0:5:90;
pat_phitheta = uv2phithetapat(pat_uv,u,v,phi,theta);
```

Plot the result.

```
H = surf(phi,theta,pat_phitheta);
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]
```